ESTIMACIÓN DE MODELOS DE COMPORTAMIENTO DE PAVIMENTOS MEDIANTE MODELOS DE REGRESIÓN NO LINEAL POR CLUSTERWISE BASADO EN ALGORITMOS META-HEURÍSTICOS

Monografía

Richard Ruiz Díaz

Francisco Javier Anacona Campo

Director: PhD. Carlos Alberto Cobos Lozada

Codirectora: PhD. Martha Eliana Mendoza Becerra

Codirector externo: PhD. Alexander Paz (University of Nevada)

Asesores: PhD. Mukesh Khadka (University of Nevada)

MSc. Cristian Arteaga (University of Nevada)

UNIVERSIDAD DEL CAUCA

FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES

PROGRAMA DE INGENIERIA DE SISTEMAS

GRUPO DE I+D EN TECNOLOGÍAS DE LA INFORMACIÓN

LÍNEA DE INVESTIGACIÓN: GESTIÓN DE LA INFORMACIÓN - MINERÍA DE DATOS

POPAYÁN- CAUCA

JULIO DE 2018

TABLA DE CONTENIDO

INDICE DE FIGURAS

CAPÍTULO 1

1 INTRODUCCIÓN

1.1 PLANTEAMIENTO DEL PROBLEMA

La gestión apropiada de los pavimentos es clave para disminuir costos en su reparación y mantenimiento, además de aumentar la vida útil de los mismos. El deterioro del pavimento depende del paso del tiempo, las condiciones de tráfico, clima, edad, entre otros factores [1]. Una herramienta útil para la adecuada gestión del pavimento, son los modelos de predicción del comportamiento del pavimento, los cuales deben ser precisos y atemperados a las diferentes condiciones a las que están expuestas las rutas viales [2].

Los modelos de comportamiento del pavimento son elementales para la gestión de las vías, pronostican el deterioro y alertan con anticipación cuales rutas deben ser intervenidas, además de indicar el tipo procedimiento a realizar [3]. Existen diferentes métodos de modelamiento empíricos y mecanicistas. La mayoría de los modelos que se reportan en la literatura, son simples, incluyen sólo unas pocas variables explicativas y no incorporan los efectos del mantenimiento y de la reparación, por tal razón, el desempeño de la mayoría de estos modelos según la literatura no ha sido satisfactorio. En otros trabajos se han formulado modelos más complejos, sin embargo las estimaciones de la importancia de las variables, en algunos casos, tienen resultados opuestos a los esperados [4]; Además, se requiere la intensiva participación de expertos que ayuden a discriminar variables de poca relevancia con el fin de mejorar la predicción del PSI (Pavement Serviceability Index), la cual es la medida reportada en la literatura para definir el comportamiento del pavimento.

El PSI depende de dos parámetros fundamentales que son [5]: fiabilidad y vida esperada. El primero es el promedio del servicio de un segmento de carretera basado en un nivel

mínimo de aceptabilidad y el segundo es el parámetro que resulta ser el mejor para determinar el comportamiento [6]. El PSI es un valor que se calcula por segmentos de carretera, ya que cada segmento a pesar de ser construido con el mismo material y en fechas similares, puede estar expuesto a diferentes condiciones y haber sido sujeto a diferente número de reparaciones o mantenimientos [7]. Por otro lado, determinar el PSI en forma individual es un proceso complejo y costoso, por esto, se han desarrollado diferentes investigaciones, donde se agrupan segmentos con factores y condiciones similares.

Basado en lo anterior, en la literatura se reportan algunos métodos de agrupación (Clustering) para explicar a través de modelos lineales y no lineales el comportamiento del pavimento, como por ejemplo el uso de Regresión Lineal por Clusterwise (CLR) [8] y el uso de algoritmos meta-heurísticos para ajustar los valores de los modelos de predicción con el fin de hacerlos más precisos [9].

Dentro de los modelos Clusterwise, se resaltan los recientes resultados de Khadka [10], en los cuales se adapta un algoritmo de Recocido Simulado (Algoritmo meta-heurístico de estado simple muy popular en problemas de optimización compleja)[11], para definir los grupos de segmentos similares, además a cada grupo se le define el modelo lineal o no lineal que mejor predice el valor de su PSI. En este trabajo no se evaluaron otras alternativas de algoritmos meta-heurísticos poblacionales, que en muchos problemas hoy en día obtienen mejores soluciones, como por ejemplo Global-best Harmony Search (GHS) [12], Differential Evolution (DE) [13] y Particle Swarm Optimization (PSO) [14]. Además, sólo trabajaron con un modelo lineal y uno no lineal.

Teniendo en cuenta, que de los No-Free-Lunch Theorems (NFLT), se puede establecer que para un problema específico de optimización, no se conoce cuál de las diferentes meta-heurísticas disponibles puede resolver un problema específico de la mejor manera y que esto sólo se puede definir de forma experimental [15]. Además, que los algoritmos meméticos hoy día, son el estado del arte en la solución de diversos problemas de optimización continua, discreta y binaria [16, 17]. Se considera necesario explorar otras técnicas como lo mencionan en el trabajo futuro de Khadka [9], para obtener soluciones optimizadas que sean mejores a las disponibles hoy en el estado del arte [14].

En este proyecto se formuló la siguiente pregunta de investigación: ¿Cómo es posible obtener mejores modelos de predicción del comportamiento de pavimentos, mediante modelos de regresión lineal o no lineal por clusterwise, basado en algoritmos meméticos? Buscando acotar el alcance del trabajo, se evaluaron tres meta-heurísticas poblacionales (GHS, ED, PSO), se convirtieron en propuestas meméticas, incluyendo un optimizador local, basado en Ascenso de Colina [18] y se compararon entre sí y con el trabajo realizado por Khadka.

1.2 APORTES

La contribución de este proyecto desde la perspectiva de investigación se centró en generar nuevo conocimiento enfocado en la optimización de modelos de regresión no lineal por clusterwise utilizando algoritmos meta-heurísticos (GHS, ED, PSO) y meméticos, resultados de usar como búsqueda local el Ascenso a la Colina (HC) y conocimiento del problema. Se definió cuál de los algoritmos (3 meta-heurísticos y 3 meméticos) genera mejores modelos de predicción y si las propuestas son mejores a los del estado del arte.

Desde la perspectiva de innovación, en este proyecto se implementaron los tres algoritmos meta-heurísticos (GBHS, ED y PSO) y los tres meméticos (GBHS-M, ED-M y PSO-M), se evaluaron y se compararon sobre modelos de gestión de pavimentos con CLR. Estos algoritmos se implementaron en R y quedan disponibles a la comunidad académica, científica e industrial que los requiera.

Estas estrategias de solución son nuevas para la comunidad y al dejarlas disponibles se espera que tengan un impacto global en las actividades de gestión de pavimentos que en Colombia normalmente la desarrollan los ingenieros civiles.

1.3 OBJETIVOS

A continuación, se presentan los objetivos como fueron aprobados en el anteproyecto por parte del Consejo de Facultad de la Facultad de Ingeniería Electrónica y Telecomunicaciones de la Universidad del Cauca.

1.3.1 OBJETIVO GENERAL

Proponer un algoritmo meta-heurístico para optimizar la estimación de modelos de comportamiento de pavimentos, mediante modelos de regresión no lineal por

clusterwise, buscando obtener modelos más precisos que los disponibles en la actualidad.

1.3.2 OBJETIVOS ESPECÍFICOS

- Definir un modelo de regresión no lineal de comportamiento de pavimentos, por clusterwise, que contemple variables independientes, continuas y categóricas, para que sea más sencillo (reduzca el número de variables) y facilite la interpretación del mismo.
- Adaptar tres algoritmos meta-heurísticos (Evolución Diferencial, Optimización por Enjambre de Partículas y la Mejor Búsqueda Armónica Global) que permitan realizar el proceso de conformación de los grupos (clúster) y la definición de los modelos de regresión de cada grupo, buscando obtener una mejor precisión en los modelos obtenidos y comparar los resultados con el estado del arte (Recocido Simulado).
- Definir el mejor algoritmo memético, resultante de incluir una estrategia de optimización local (Ascenso de Colina) con conocimiento del problema a las soluciones propuestas y comparar los resultados basado en la calidad (menor suma de error cuadrado medio) de los resultados obtenidos.

1.4 RESULTADOS OBTENIDOS

A continuación, se resumen los principales resultados del presente trabajo de investigación:

- Monografía del trabajo de investigación, corresponde al presente documento que contiene el estado del arte sobre el problema que se abordó, junto con la explicación detallada de los algoritmos propuestos y utilizados en el proceso de experimentación. Luego se presenta el análisis de los resultados obtenidos por los algoritmos meta-heurísticos propuestos junto con la comparación entre ellos y el estado del arte. Terminando con las conclusiones y el trabajo futuro que el grupo de investigación espera desarrollar en el tema.
- Scripts en R con la implementación de los algoritmos junto con su respectiva documentación y manual instructivo para la ejecución y replicación de los experimentos.
- Artículo “Estimación de modelos de comportamiento de pavimentos mediante modelos de regresión no lineal por clusterwise basado en algoritmos meta-heurísticos” que resume la investigación y los resultados obtenidos.

1.5 ESTRUCTURA DE LA MONOGRAFÍA

A continuación, se describe de manera general el contenido y organización de la presente monografía.

Capítulo 1: introducción: Corresponde al presente capítulo, que presenta una introducción al tema de investigación, la pregunta de investigación, los aportes del trabajo, los objetivos (general y específicos) aprobados, el resumen de los resultados obtenidos y finalmente la organización de cada capítulo de la monografía.

Capítulo 2: contexto teórico y estado del arte: En la primera parte de este capítulo se presenta por un lado la descripción y explicación en detalle de los conceptos que se usaron en la investigación. Finalmente se muestran las investigaciones que sobresalen en el estado del arte relacionadas con el problema de investigación.

Capítulo 3: propuesta: En este capítulo se hace una descripción detallada de la solución al problema de los modelos de regresión no lineal para la gestión de pavimentos por clusterwise usando las meta-heurísticas de búsqueda global PSO, GBHS y ED, con sus variantes meméticas agregando el algoritmo de búsqueda local HC.

Capítulo 4: experimentación: En este capítulo se define el escenario de experimentación y luego se presentan y analizan los resultados de la ejecución de la meta-heurísticas con el modelo de regresión no lineal para la gestión de pavimentos.

Capítulo 5: conclusiones y trabajo futuro: En este capítulo en su primera parte se presentan las conclusiones a las que se llegó una vez terminado el trabajo de investigación, en la segunda parte se mencionan las posibles mejoras que se pueden agregar al trabajo realizado.

Capítulo 6: bibliografía: En este capítulo se listan las referencias de libros, artículos y documentos consultados durante la realización del proyecto.

CAPÍTULO 2

2 CONTEXTO TEÓRICO Y ESTADO DEL ARTE

A continuación, se presentan conceptos utilizados en el proyecto con el fin de facilitar el entendimiento de este.

2.1 MODELOS DE COMPORTAMIENTO DE PAVIMENTOS

En modelos de comportamiento de pavimentos (Pavement Performance Models, PPM)[19], se define que los pronósticos de las condiciones de pavimentos, deben contar con bases de datos que contengan una cantidad mínima de muestras disponibles de un tramo de carretera (segmento). Estos modelos predicen el grado del índice de servicio actual (PSI) [20, 21], que es una medida temporal, que se le asigna a un tramo de carretera cada año, determinando el comportamiento del pavimento que por regla maneja un rango de 0 a 5, donde cero indica mal estado y cinco las condiciones óptimas del segmento. Dependiendo de esta medición se determina el mantenimiento o rehabilitación del pavimento [22], el cual se define como un proceso rutinario, preventivo o reactivo, con actividades consistentes en rellenar grietas, parchear baches y otras técnicas aplicables, todo para mejorar la capacidad estructural de los pavimentos.

2.1.1 MODELOS DE GESTIÓN DE PAVIMENTOS

El uso eficiente de los materiales y el tiempo, es lo que conduce a la optimización de costos, objetivo principal de los sistemas de gestión de pavimentos (Pavement Management Systems, PMS) [23]. Los PMS ayudan a tomar decisiones en las reparaciones y mantenimiento que se debe utilizar, donde y cuando aplicarlo, para aprovechar al máximo los recursos disponibles. Algunos de esos PMS usan modelos de regresión lineal por clusterwise [8] para definir el PSI y con ello tomar las mejores decisiones posibles.

2.1.2 THE PRESENT SERVICEABILITY INDEX

En los modelos de comportamiento de pavimentos se toman un número de observaciones (segmentos de malla vial), se agrupan de la forma más homogénea

posible (los segmentos de cada grupo son lo más parecidos entre sí), luego se calcula un modelo de regresión para predecir el PSI de cada grupo (esto se hace buscando minimizar la suma de los errores cuadráticos entre los datos reales de cada grupo y los predichos por el modelo), y cuando se obtiene la mejor configuración de grupos y modelos asociados a los grupos se termina el proceso. Para lograr esto, se han usado algoritmos meta-heurísticos, como lo realizan en su trabajo W. S. DeSarbo, R. L. Oliver, and A. Rangaswamy [9], donde usan un algoritmo de estado simple y obtienen buenos resultados.

2.2 ALGORITMOS META-HEURÍSTICOS

Las meta-heurísticas han tenido un gran avance a lo largo de los años, debido a que han dado de forma exitosa solución a diversos problemas de optimización combinatoria con diferentes grados de dificultad. Las meta-heurísticas tienen diferentes enfoques como lo son: de estado simple como recocido simulado y poblacionales como los algoritmos genéticos y optimización basada en enjambre de partículas. Incluyen conceptos de solución inteligente a problemas con la ayuda de conceptos de las matemáticas, la estadística, entre otras [24].

Algunos de los algoritmos se basan en métodos numéricos de programación lineal y no lineal, por lo general buscan maximizar o minimizar una función objetivo que mide la calidad de los resultados, con el fin de dar la mejor solución posible a un problema de optimización manteniendo un costo computacional razonable, pero sin garantizar la optimalidad de este. Si existen diversos óptimos locales (una solución que puede ser buena pero no la mejor a nivel global) los algoritmos deben aplicar técnicas que verifiquen, sea la mejor de todo el entorno global al problema y evitar quedar atrapado en dichas soluciones. Es por ello que las meta-heurísticas son estrategias ingeniosas que mejoran los procedimientos heurísticos [25].

2.3 ALGORITMOS META-HEURÍSTICOS POBLACIONALES

Algunos de los algoritmos no son capaces de proporcionar soluciones a un problema de optimización con un espacio de búsqueda de alta dimensionalidad. En estos espacios, la búsqueda crece exponencialmente y hacerlo de forma exhaustiva no es una opción viable en la mayoría de los casos. Además de usar métodos clásicos como los algoritmos "Greedy" [26], hacen muchas suposiciones para solucionar un problema, la validación de estas suposiciones es difícil en cada problema. Es por ello, que el utilizar las meta-heurísticas adecuadas sobre un problema determinado sirve para generar soluciones

que resuelvan apropiadamente (calidad de la solución vs el tiempo de ejecución) un problema, por ello los algoritmos basados en población son adecuados para búsquedas globales, ya que combinan la exploración global con la explotación. Existen varios tipos de algoritmos para distintos problemas, como lo son los basados en población continuos (reales), discretos y binarios.

2.3.1 OPTIMIZACIÓN POR ENJAMBRE DE PARTÍCULAS

El algoritmo de optimización por enjambre de partículas (Particle Swarm Optimization, PSO) [14], se basa en el comportamiento social de un enjambre de partículas (individuos) que explora un espacio de soluciones a un problema. Cada partícula es una posible solución y se mueve hacia diferentes posiciones del espacio de búsqueda de las otras posibles soluciones al problema basada en su propio conocimiento (historia de posiciones visitadas) y la posición de la mejor partícula del enjambre que corresponde al conocimiento del enjambre. Después de un tiempo o de número de iteraciones ejecutadas, el algoritmo retorna la mejor solución encontrada por el enjambre.

2.3.2 LA MEJOR BÚSQUEDA ARMÓNICA GLOBAL

El algoritmo de la mejor búsqueda armónica global (Global-best Harmony Search, GHS) [12, 27] es un algoritmo basado en la forma como los músicos de Jazz realizan improvisación musical. Este algoritmo es poblacional y estacionario, en el cual se genera una nueva solución en cada iteración con información de la población, del mejor de la población y en forma aleatoria de los rangos de las variables (a manera de una mutación) combinando de forma adecuada formas de exploración y explotación. Uno de los puntos fuertes es su improvisación ya que no hace procesos complejos de cálculos matemáticos, lo que lo hace mucho más rápido y lo diferencia de otros que existen [28]. Su diseño le permite resolver problemas continuos, discretos y binarios.

2.3.3 EVOLUCIÓN DIFERENCIAL

El algoritmo de Evolución Diferencial (Differential Evolution, DE) [13, 29], cuenta con un proceso de optimización muy efectivo en problemas continuos. Este algoritmo se centra en la mutación de los individuos con operadores de recombinación y cruce. Es efectivo para encontrar óptimos globales para diferentes funciones de prueba. La población que se genera inicialmente debe ser aleatoria para seguir una distribución uniforme con un dominio variable.

2.3.4 RECOCIDO SIMULADO

El algoritmo de recocido simulado (Simulated Annealing), es uno de los mejores algoritmos de estado simple que resuelve problemas grandes de optimización reportados en el estado del arte. Emula el proceso físico de enfriamiento gradual que se utiliza cuando se trabaja el metal, donde la temperatura es el parámetro que rige la distancia que hay de una solución candidata a una ya existente [30], este algoritmo tiene la particularidad que acepta soluciones aún si no son mejores que la actual según disminuye la temperatura, con el fin de mejorar el proceso de exploración y no caer en óptimos locales, pero en el momento que pasa cierto umbral comienza a hacer explotación hasta que el parámetro de temperatura sea mínimo con el fin de que el algoritmo converja a la mejor solución óptima posible [31].

2.4 ALGORITMOS MEMÉTICOS

Los algoritmos meméticos se definen en la literatura como la combinación de un algoritmo de optimización local, con un algoritmo de búsqueda global [32] aplicando conocimiento del problema que permita acelerar la búsqueda en regiones donde se puedan encontrar mejores soluciones [16, 17]. Constituyen una pauta de optimización basada en la explotación sistemática de soluciones a partir de un problema que debe ser resuelto, involucrando ideas de diferentes técnicas. Estos algoritmos tienen como tema principal la hibridación de diferentes enfoques para un problema dado. Así que incluyen varios algoritmos poblaciones con algunos de estado simple y sus posibles variantes poniéndolos a competir para determinar cuál es el mejor para determinado problema. Hoy en día varias de esas hibridaciones incluyen otras meta-heurísticas que dan soluciones de alta calidad en los inicios del proceso de búsqueda. Por último el adjetivo "memético" proviene del término "meme" [33] para indicar un semejante al gen en el contexto de la evolución cultural [34].

2.5 ALGORITMOS DE BÚSQUEDA LOCAL

Son técnicas muy usadas como heurísticas para resolver problemas de optimización binaria, entera y real (continua). Estas heurísticas no dan garantía de alcanzar el óptimo ya que pueden estancarse en óptimos locales. Se usan para explotar una solución y encontrar la mejor solución local cercana o vecina [35]. En general son simples en su implementación, y computacionalmente no muy costosas [36].

2.5.1 ALGORITMO DE ASCENSO A LA COLINA

Es un algoritmo de búsqueda local para problemas de optimización, donde a partir de uno datos de entrada (solución inicial) y una función objetivo o de aptitud, intenta hallar soluciones a un problema maximizando o minimizando dicha función en un proceso iterativo hasta que se cumpla un criterio de parada. Aunque hay casos en los que la solución puede no ser la mejor optima global, posiblemente si es aceptable dentro de las restricciones de tiempo de ejecución o procesamiento computacional que se tengan. El algoritmo de ascenso realiza los siguientes pasos principales:

1. Generar una solución respetando las restricciones del problema y decir que es la mejor encontrada hasta el momento
2. Realizar una copia de la solución generada y mutarla
3. Comparar el fitness de las dos soluciones y si la solución mutada es mejor, se remplaza como la mejor solución encontrada hasta el momento
4. Repetir los pasos 2 y 3 hasta que se cumpla un criterio de parada.

2.6 ALGORITMOS DE AGRUPACIÓN

El agrupamiento o Clustering, es una tarea de la minería de datos, conocida como aprendizaje no supervisado en la inteligencia artificial, que busca agrupa de manera automática elementos de un conjunto de datos basado en alguna relación de similitud. Para realizar un buen agrupamiento se tienen en cuenta las características comunes de los elementos y así se identifica que elementos deben ir juntos y cuales deben ir separados o en distintos grupos, para lograr esto, normalmente se usan medidas de similitud como la de cosenos o de distancia como la euclidiana. Este proceso busca que los individuos de un mismo grupo o clúster sean muy semejantes entre sí y diferentes de los individuos de los grupos restantes. Para realizar un buen agrupamiento es fundamental realizar un análisis de datos para comprender el contexto del problema [37]. Los algoritmos de agrupamiento no solo se usan para organizar y clasificar datos, también se usan como una herramienta para la interpretación de los resultados de las investigaciones y la construcción de modelos [38].

2.6.1 K-MEANS

Al algoritmo de agrupamiento k-means, se basa en un enfoque de partición repetitivo, es una de las técnicas más usadas según el estado del arte y tiene como objetivo dividir n observaciones en k clústeres, donde k es el número de clústeres que se desean obtener. Cada observación se reubica iterativamente en el centroide más cercano y se ajusta la

ubicación del centroide según los miembros de cada grupo. No obstante, lo sencillo y útil que es el k-means, el rendimiento (calidad de los resultados) depende de los valores iniciales de los centroides seleccionados aleatoriamente. El algoritmo puede llegar a soluciones optimas locales y estancarse, por ello, el uso de un enfoque meta-heurístico que usa k-means permite obtener resultados de mejor calidad a pesar del mayor costo computacional en el que se incurre [39].

2.6.2 REGRESIÓN LINEAL POR CLUSTERWISE

La regresión lineal por clusterwise es un método estadístico semi-paramétrico, que permite mitigar los efectos de los errores de un modelo de predicción en los resultados de un problema , a través del estudio y análisis de la relación que hay entre las variables [40]. Cuando se tiene una base de datos de información de un problema en específico, se quiere saber cuál es la relación que tienen estos datos y es necesario comprender el rol de cada variable explicativa y pronosticar los posibles resultados de los procesos que se ejecuten con cada uno. Con el uso de regresión lineal por clusterwise, se pueden lograr esos resultados, definiendo el número de grupos de observaciones de tal manera que la suma de los errores cuadrados en la aproximación lineal de cada uno de los grupos es la mínima posible [40].

2.7 RESTRICTED GROWTH STRINGS

Una cadena de crecimiento restringido o RGS [41], es una cadena de n posiciones enteras donde la primer posición tiene un valor de 1 y cada valor ubicado en la posición i+1 es igual al valor en la posición i o supéralo como máximo en uno. Las cadenas se basan en la **in-Ecuación (1)** para el crecimiento.

$$a[i+1] \leq 1 + \max(a[1], a[2], \ldots, a[i]) \text{ Con } a[1] = 1 \quad (1)$$

Aplicadas al clustering, las RGS ayudan a reducir el espacio de búsqueda de las soluciones. La **Figura 1** muestra los subconjuntos que se generan a partir de un conjunto 4 elementos (N = 4) con su correspondiente cadena en representación RGS. En la figura el subconjunto {1234} se representa por la cadena {0000} indicando que todos los elementos pertenecen al grupo 0 (bloque 0) y el subconjunto {123|4} indicando en este que los elementos 1,2 y 3 pertenecen al grupo 0 (bloque 0) y el elemento 4 pertenece al grupo 1 (bloque 1).

El elemento i de la cadena resultante no puede estar incluido en un grupo cuyo índice sea mayor en una unidad del índice mayor de los grupos de los elementos anteriores. Las RGS son fundamentales para este trabajo ya que se puede representar el conjunto de particiones con una restricción de crecimiento, los grupos están ordenados de tal manera que se segura que cadenas como 0011 o {12.34} y 1100 o {34.12} no se consideren diferentes lo que optimiza la búsqueda de soluciones, asegurando la unicidad de cada solución. Si se usa una cadena sin la restricción de crecimiento el número total de cadenas requeridas para buscar con k = 1, 2, 3 y 4 grupos el número de posibles cadenas es 256 (4^4) que es mucho mayor que 15 (2^4-1), las requeridas con RGS para este ejemplo de 4 elementos.

Subconjuntos	***Cadena RGS***	***K Grupos***	***Cadenas RGS requeridas***
{1234}	0000	K = 1	1
{123,4}	0001	K = 2	7
{124,3}	0010		
{134,2}	0100		
{1,234}	0111		
{12,34}	0011		
{13,24}	0101		
{14,23}	0110		
{1,2,34}	0122	K = 3	6
{1,24,3}	0121		
{1,23,4}	0112		
{14,2,3}	0120		
{13,2,4}	0102		
{12,3,4}	0012		
{1,2,3,4}	0123	K = 4	1
Cadenas totales requeridas según la representación:			15 = 2^4-1

Figura 1. Ejemplos de Conjuntos RGS para clustering.

3 ESTADO DEL ARTE

En esta sección, se presentan las investigaciones más representativas relacionadas con modelos de regresión lineales y no lineales para gestión de pavimentos por clusterwise, con el objetivo de conocer las técnicas o procesos que se han empleado a lo largo de los últimos años.

En la revisión de la literatura se encuentra que la Regresión Lineal Por Clusterwise es una técnica que se ha aplicado en diferentes campos incluido la gestión de pavimentos. Las propuestas de modelos de gestión de pavimentos que incluyen técnicas meta-

heurísticas no son muy comunes. A continuación, se describen los artículos de mayor relevancia encontrados en la revisión de la literatura.

A simulated annealing methodology for clusterwise linear regression [9] (1989): Los autores de este trabajo, propusieron una metodología que agrupa las observaciones (segmentos viales) a un número preestablecido de grupos, con el fin de estimar los coeficientes de las funciones de regresión para optimizar una función objetivo en común. Describen en su propuesta un recocido simulado que realiza agrupamientos superpuestos o no superpuestos, variables dependientes, univariables o multivariables y restricciones impuestas a cada grupo. Con el recocido simulado garantizan la búsqueda de diversos tamaños de grupos y las diferentes especificaciones del modelo con lo que computacionalmente es más eficiente aplicando el método de Späth [42]. Finalmente, recomienda usar regresiones no lineales y algoritmos combinatorios más rápidos y eficientes, debido a que el algoritmo se queda atrapado en óptimos locales.

Incremental nonlinear model for predicting pavement serviceability [43] (2003): En este trabajo propusieron un modelo recursivo no lineal para la predicción del comportamiento del pavimento en función de las características del tráfico, propiedades estructurales y las condiciones ambientales. El modelo destaca algunas de las ventajas de relajar la restricción lineal que se suele colocar en los modelos de comportamiento de pavimentos (PPM)[19]. Esos modelos mejoran la representación del deterioro del pavimento, luego los parámetros estimados son imparciales, debido a una especificación del contexto del problema y uso de técnicas estadísticas sólidas. Finalmente, el error estándar de predicción se reduce a la mitad del modelo lineal equivalente que ya existe. Esa mejora tiene importantes implicaciones económicas en el contexto de la gestión del pavimento (PMS)[23].

Pavement condition prediction using clusterwise regression [44] (2006): En este trabajo propusieron un modelo de regresión por grupos que ajusta los datos a más de una curva, para modelar el deterioro de la condición del pavimento. Realizaron una modificación al modelo por la pertenencia de una muestra a un grupo que se estima con el concepto de conjuntos difusos. El número de incógnitas en el modelo modificado se reduce significativamente. El modelo se extendió posteriormente, para ser aplicable a casos en los que se utilizaron más de dos grupos o ecuaciones no lineales. Se propuso un procedimiento para predecir el grado de calificación del pavimento. El método de regresión de mínimos cuadrados ordinarios se empleó por primera vez para determinar

la curva de predicción de PMS para un grupo de pavimentos. Los resultados mostraron que los procedimientos propuestos usando el método de regresión por grupos modificados, dieron un menor error de predicción y así mejoraron la precisión de la curva ajustada. Los autores recomiendan este modelo en los casos en que las familias de pavimentos no estén bien definidas. El número óptimo de grupos puede determinarse mediante la inspección del cambio de la suma de errores cuadráticos (SSE) [45], con respecto al cambio del número de grupos. La exactitud del modelo puede justificarse mediante la comparación de predicciones previstas con las predicciones observadas.

Development of linear mixed effects models for predicting individual pavement conditions [46] (2007) : En este trabajo el autor propone utilizar un modelo de efectos mixtos lineales para predecir las condiciones futuras de una sección de pavimento especifica mediante una combinación ponderada de la tendencia media del deterioro y las condiciones pasadas del pavimento. Los pesos relativos se determinan por el número de mediciones en su historial de muestras disponibles y el grado de variaciones de las condiciones pasadas del pavimento que fueron medidas. Los resultados de este trabajo muestran una precisión significativamente mayor en la predicción de condiciones específicas de pavimento comparada con los métodos de ajuste existentes que se utilizaron en la última medición de condición disponible de pavimento para ajustar la predicción en tendencia familiar. Lo que implica que estos resultados pueden utilizarse para predicción de condiciones de pavimento a nivel de proyectos u otros tipos de predicción de condiciones de infraestructura.

Algorithms for generalized cluster-wise linear regression [47] (2017): En este trabajo proponen un enfoque basado en programación matemática, con un algoritmo meta-heurístico que agrupa conjuntos predefinidos de entidades, incluyendo técnicas de minería de datos, adaptando un enfoque en dos etapas que realiza agrupamiento y regresión, resolviendo el CLR [8] generalizado. La propuesta se evaluó en un problema de agrupamiento empleado para pronósticos y tendencias de datos. El algoritmo genético más prometedor es capaz de generar agrupamientos, con patrones estacionales distintivos de manera eficaz y eficiente, el algoritmo de dos etapas produce grupos muy rápido, pero con valores de objetos más grandes que el algoritmo genético. Aunque, en este trabajo el contexto del problema no involucra temas sobre gestión de pavimentos, es importante, debido a que fusiona algoritmos meta-heurísticos con técnicas de minería de datos y el método de regresión linear por clusterwise.

Finalmente, a partir de los registros de datos (muestras recopiladas durante doce años) proporcionados por el Departamento de Transporte de Nevada (NDOT) que son reconocidas por la comunidad científica, Khadka [10] (2017) propone un modelo de regresión por clusterwise (CLR). Este trabajo es actualmente el estado del arte y corresponde a una tesis de doctorado donde se propone un modelo lineal y uno no lineal, un algoritmo basado en recocido simulado y una formulación matemática que generaliza el problema [9]. La función objetivo que usa el algoritmo es el criterio de información bayesiano (BIC), explorando las combinaciones posibles de variables explicativas en cada agrupación de segmentos y de esta forma selecciona la mejor especificación de los modelos de cada grupo. Las exactitudes predictivas de los modelos resultantes se evaluaron utilizando errores cuadráticos medios, errores cuadráticos medios normalizados y la media de los errores absolutos. Los resultados de este trabajo evidencian que los modelos no lineales fueron más precisos que los modelos lineales en la estimación del PSI [6].

CAPÍTULO 3

4 PROPUESTA

4.1 INTRODUCCIÓN

Este trabajo se basa en la investigación realizada por Mukesh Khadka en su tesis doctoral [10], en la cual se plantearon modelos de regresión lineal y no lineal, para modelar el comportamiento de pavimentos por clusterwise, en el cual propone explorar otras meta-heurísticas en el campo de la regresión no lineal, que actualmente son el estado del arte. En este trabajo se optó por usar el modelo matemático y la función objetivo planteadas por Khadka, y la aplicación de las meta-heurísticas PSO, GBHS y ED, que no se han utilizado en este contexto, además de sus versiones meméticas, resultado de su hibridación con Hill Climbing.

4.2 TRABAJO BASE

En la propuesta de Khadka se cuenta con una base de datos con muestras de pavimentos, en las cuales se tienen diferentes tipos de variables explicativas continuas y categóricas, utilizadas para realizar modelos lineales y no lineales de comportamiento de pavimentos. Adaptó un algoritmo meta-heurístico de estado simple, recocido simulado, minimizando la cantidad optima de grupos con BIC (criterio de información bayesiano) como función objetivo. Con el BIC buscan contrarrestar la convergencia de las soluciones a soluciones con mayor número de grupos (que afecta a la Suma del Error Cuadrático o SSE) y logra un equilibrio entre la bondad de ajuste y la complejidad del modelo. Finalmente, el proceso selecciona el mejor modelo de cada grupo al explorar todas las posibles combinaciones de las variables de entrada, lo que contribuye a reducir el error de estimación en los modelos de gestión de pavimentos y a calcular automáticamente el número óptimo de clusters.

4.3 DATA SET

Periódicamente los departamentos de transporte de algunos países como Estados Unidos recolectan muestras de pavimento, esto con el fin de obtener información detallada del estado de cada una de sus carreteras y según las condiciones de estas

tomar decisiones. En el presente trabajo se toma como datos de investigación, los datos proporcionados por el Departamento de Transporte de Nevada. Estos datos tienen 14.638 muestras de pavimento de ese lugar, muestras recolectadas desde el año 2001 al 2010, con las que se desarrolla el modelo de gestión de pavimentos en el trabajo base y la presente investigación. A continuación, se muestra el procesamiento que se le da al dataset con el fin de adaptarlo a las necesidades del problema.

4.3.1 TRATAMIENTOS AL DATA SET

La información del dataset que se maneja es de 14.638 muestras. La cantidad de variables que se utilizan es de 17, de las cuales 10 son explicativas continuas y 4 categóricas. A continuación, en la **Figura 2**, se muestran 8 observaciones que corresponden a 2 tramos diferentes de carretera.

	sample_id	time_period	psi	age	aadt	trucks	elevation	precip	min_temp	max_temp	wet_days	freeze_thaw	rut_depth	number_of_lanes	sys_id	f_class	category
1	1	4	3.82	0	725	20	4750	8.25	33	65	45	176	0.090	1	3	5	3
2	1	4	3.73	1	825	20	4750	8.25	33	65	45	176	0.080	1	3	5	4
3	1	4	3.71	2	950	19	4750	6.65	36	67	41	154	0.080	1	3	5	4
4	1	4	3.62	3	950	20	4750	6.65	36	67	41	154	0.090	1	3	5	4
5	2	4	3.96	0	725	20	4750	6.65	36	67	41	154	0.010	1	3	5	3
6	2	4	3.88	1	825	20	4750	6.65	36	67	41	154	0.010	1	3	5	4
7	2	4	3.86	2	950	19	4750	6.65	36	67	41	154	0.010	1	3	5	4
8	2	4	3.53	3	950	20	4750	6.65	36	67	41	154	0.010	1	3	5	4

Figura 2. Representación de segmentos en el dataset

Un tramo de carretera se interpreta como un segmento de carretera, esas observaciones son el histórico de la recolección de dos segmentos, donde:

- *SAMPLE_ID*: Significa el tramo de pavimento del que se obtiene la muestra.
- *TIME_PERIOD:* Hace referencia al periodo actual.
- *PSI:* Es la variable estándar que representa la medida del estado actual de la carretera, valor que con el tiempo varia y disminuye en cada periodo.

Variables Explicativas:

- *AGE*: Representa la edad o época en la que la carretera tuvo una intervención por reparación o mantenimiento, como lo muestra la figura.
- *AADT:* Representa el tráfico promedio a la que está sometida una carretera en un día en una de las direcciones.
- *TRUCKS:* Representa la medida de tráfico promedio de camiones a la que está expuesta la carretera
- *ELEVATION:* Es la elevación en el punto medio de un segmento en metros.

- *PRECIP:* Representa la precipitación media anual con una medida de (cm/año).
- *MIN_TEMP:* Representa la temperatura media anual mínima del aire tomada en grados centígrados.
- *MAX_TEMP:* Representa la temperatura media anual máxima del aire tomada en grados centígrados.
- *WET_DAYS:* Representa el total de número de días húmedos en el transcurso del año
- *FREEZE_THAW:* Representa el número total de ciclos de congelación o descongelación que experimenta un pavimento en el transcurso del año.
- *RUT_DEPTH:* Representa la profundidad del aboyamiento medio de una carretera.

Variables Categóricas:

- *NUMBER_OF_LANES:* Representa la variable dummy para el número de carriles que tiene una carretera.
- *SYS_ID:* Representa la variable dummy para clasificar si el sistema pertenece a NHS (Sistema Nacional de Carretera), STP (Programa de Transporte de Superficie) o IR (Ruta Interestatal).
- *F_CLASS:* Representa la variable dummy para un segmento clasificado como funcional.
- *CATEGORY:* Representa la variable dummy para categorizar la priorización de un segmento de carretera, utilizando factores como el volumen del tráfico y al frecuencia de actividades de manteamiento y reparación.

4.3.2 REPRESENTACIÓN DE UN SEGMENTO

Un segmento, es un conjunto de muestras de pavimento tomados de un mismo tramo de carretera, la base de datos con la que se experimenta en esta investigación reúne 3.674 segmentos, a continuación, en la **Figura 3** se muestra la estructura de un segmento:

	sample_id	time_period	psi	age	aadt	trucks	elevation	precip	min_temp	max_temp	wet_days	freeze_thaw	rut_depth	number_of_lanes	sys_id	f_class	category
1	1	4	3.82	0	725	20	4750	8.25	33	65	45	176	0.090	1	3	5	3
2	1	4	3.73	1	825	20	4750	8.25	33	65	45	176	0.080	1	3	5	4
3	1	4	3.71	2	950	19	4750	6.65	36	67	41	154	0.080	1	3	5	4
4	1	4	3.62	3	950	20	4750	6.65	36	67	41	154	0.090	1	3	5	4

Figura 3. Representación de un segmento

4.4 FUNCIÓN OBJETIVO

La función objetivo a minimizar que se utilizó para el trabajo de investigación es el Criterio de Información Bayesiano (BIC) expresada en la **Ecuación (2)**.

$$Min\ BIC = O + O * \ln(2\pi) + O * \ln\left(\frac{SSE}{O}\right) + (\delta + K - 1) * \ln(n) \quad (2)$$

Donde:

- O = Número total de observaciones.
- *SSE* = Suma de errores cuadráticos.
- δ = Número total de variables explicativas significativas.
- K = Número óptimo de clusters
- n = Número mínimo de observaciones requeridas en un cluster.

4.5 PREPROCESAMIENTO

El preprocesamiento tiene como finalidad el preparar los datos para que los algoritmos de agrupamiento funcionen correctamente. A continuación, se explica de manera detallada los algoritmos utilizados para este procedimiento.

4.5.1 NORMALIZAR DATOS

Es necesario antes de la ejecución de los algoritmos propuestos, normalizar lo campos continuos de la base de datos para luego poder usar adecuadamente k-means. El método utilizado es *Min-Max,* que convierte los valores de cada columna en un rango de [0 - 1] estos datos son guardados temporalmente mientras termina la ejecución de k-means.

$$v' = \frac{\mathrm{v} - \min}{\max - min} \quad (3)$$

Donde:

- min es el menor valor de la columna
- max es el mayor valor de la columna
- v es el valor actual del registro en la columna que se está normalizando
- v' es el valor normalizado en el rango 0 a 1.

4.5.2 TRANSFORMACIÓN LOGARÍTMICA

El estado del arte menciona, que se obtienen mejores resultados utilizando regresiones no lineales en el proceso de clusterwise para la gestión de pavimentos, por ello se hace una transformación logarítmica a los datos continuos, de esta manera se puede realizar el proceso no lineal con técnicas de regresión lineal, como se muestra en la **Figura 4**. Las variables *simple_id* y *time_period*, no se transforman debido a que no hacen parte de las variables explicativas.

sample_id	time_period	psi	age	aadt	trucks	elevation	precip	min_temp	max_temp	wet_days	freeze_thaw	rut_depth
1	4	1.3402504	0.0000000	6.586172	2.995732	8.465900	2.110213	3.496508	4.174387	3.806662	5.170484	-2.407946
1	4	1.3164082	0.0000000	6.715383	2.995732	8.465900	2.110213	3.496508	4.174387	3.806662	5.170484	-2.525729
1	4	1.3110319	0.6931472	6.856462	2.944439	8.465900	1.894617	3.583519	4.204693	3.713572	5.036953	-2.525729
1	4	1.2864740	1.0986123	6.856462	2.995732	8.465900	1.894617	3.583519	4.204693	3.713572	5.036953	-2.407946

Figura 4. Transformación logarítmica de las variables de un segmento

4.6 CREAR POBLACIÓN

A continuación, se explica cómo se crea una población de manera general para los algoritmos PSO, GBHS y ED, este método es el mismo en todos los casos. Una población está compuesta por “n” número de soluciones, cada solución tiene una estructura diferente dependiendo de la meta-heurística utilizada, si es PSO, la solución pasa a llamarse individuo y va a contener la posición actual, el BIC de esta posición, el mejor histórico y el BIC del mejor histórico, si la meta-heurística utilizada es GBHS entonces la solución pasa a llamarse armonía, esta va a contener un conjunto de tonos (similar a la posición actual en PSO) y el BIC del conjunto de tonos (posición), además de que se mantiene ordenada de menor a mayor respecto al BIC. De la misma forma cuando la meta-heurística es ED, la solución se va a llamar individuo y cada individuo va a tener su posición y el BIC de su posición. A continuación, se explica el proceso para crear las soluciones.

4.6.1 CREAR UNA SOLUCIÓN

Ya que una población está compuesta de soluciones, se debe definir el proceso para crear una solución. Dicho proceso incluye los siguientes pasos:

1. Generar clusters aleatorios.
2. Aplicar k-means sobre el resultado del paso uno.
3. Aplicar RGS sobre el resultado del paso dos.
4. Obtener fitness del resultado del paso tres.
5. Calcular el BIC del resultado del paso cuatro.

A continuación, se explican los algoritmos involucrados en este proceso.

4.6.1.1 GENERAR CLUSTERS ALEATORIOS

A partir de la base generada por Khadka, donde se realiza una inicialización al azar poco balanceada pero ajustándose a la restricción de dejar cierto número mínimo de segmentos por grupo (para hacer que los modelos de regresión de cada cluster tengan suficientes datos), se opta por proponer una forma de mejorar el punto de partida del proceso de clustering, donde primero se elimina la restricción en el número mínimo de segmentos por cluster y se propone otro método para mantener el equilibrio entre el número de elementos en los grupos haciendo la repartición de individuos con el algoritmo *Generar barajas aleatorias* de igual tamaño, haciéndolo similar a como un dealer reparte sus cartas en un casino a los k jugadores, en este caso, k grupos. Por ejemplo, con los 3674 segmentos y un número de clusters 4, se reparten los individuos de forma equitativa de modo que el resultado serán dos grupos de 918 y dos grupos de 919 segmentos. La **Figura 5** presenta el pseudocódigo de esta función.

El algoritmo de generar barajas aleatorias tiene como finalidad crear el valor de índice de identidad o pertenencia a un cluster por cada segmento, con el fin de equilibrar la cantidad de miembros por grupo. Como se indica en la línea 3, se crea un array vacío con la cantidad de elementos a utilizar (número total de segmentos), luego de la línea 4 a 9 se crea un ciclo que va hasta el tamaño de los segmentos. En la línea 5 se asigna el índice al cual pertenece el segmento al array creado en la línea 3, se aumenta en 1 el índice para que la repartición sea ordenada, en caso de llegar al límite donde el índice sea igual al número de segmentos este se reinicia y se iguala a 1, para iniciar de nuevo la repartición ordenada. Una vez se agregan todos los segmentos a los grupos se sale del ciclo como lo indica la línea 10. En la línea 11, el método inicializar clusters se crea un vector vacío con un tamaño igual al número de segmentos. En línea 12, se generan todos los valores de manera aleatoria entre el rango de uno al tamaño de los segmentos esto con el fin generar todas las posiciones de los segmentos, de la línea 13 a la 15 se realiza un ciclo for para poder ingresar a cada cluster en el vector de las posiciones los segmentos. Finalmente al terminar el ciclo se retornan todos los clusters ya con miembros.

Algoritmo Generar Barajas Aleatorias

```
1:  Function generar_barajas_aleatorias (numero_clusters, tamaño_segmentos)
2:      j = 1
3:      Segmentos <- inicializar_segmentos(tamaño_segmentos)
4:      for i to tamaño_segmentos do
5:              segmentos[i] <- j
6:              j <- j+1
7:              if j == tamaño_segmentos+1 then
8:                      j <- 1
9:              end if
10:     end for
11:     clusters <- inicializar_clusters (tamaño_segmentos)
12:     posiciones <- generar_posiciones (tamaño_segmentos)
13:     for i to tamaño_segmentos do
14:             cluster[posiciones[i]] <- segmentos[i]
15:     end for
16:     return (clusters)
17: end Function
```

Figura 5. Pseudocódigo del algoritmo para Generar Barajas Aleatorias

4.6.1.2 K-MEANS

En este trabajo se aplica k-means a las soluciones generadas por “generar barajas aleatorias”, para hacer un pre-agrupamiento de los segmentos antes de pasarlos a los diferentes algoritmos meta-heurísticos y meméticos. A este k-means se le realizó una modificación en la cual se actualizan los centroides después de que se mueve cada segmento, no al finalizar la asignación de todos los segmentos, lo que permitió obtener mejores resultados.

Al inicio del proceso se obtienen los centroides para cada grupo, esto se hace sacando el promedio de las variables explicativas y la moda en las variables categóricas. Luego, se entra en un ciclo iterativo que tiene un límite máximo de quince iteraciones con el fin de asegurar un buen agrupamiento, pero además cuando no se hacen movimientos de segmentos también se sale del bucle sin importar que aún no se completen las quince iteraciones establecidas, encontrando el mejor (óptimo local) grupo para cada segmento basado en la menor distancia euclidiana al centroide. Después se realiza el correspondiente cambio de grupo del segmento actual y se actualizan los centroides nuevamente, como se mencionó previamente, esta actualización de centroides es un cambio con respecto al k-means clásico ya que se obtuvieron mejores agrupaciones con esta modificación. Como se puede dar el caso que un grupo se quede sin segmentos, el algoritmo balancea la solución en cada grupo que tenga cero elementos como se ve en la **Figura 6**, el proceso de balanceo consiste en buscar el grupo con mayor número de

elementos (ver **Figura 7**) y luego dividirlo en dos grupos (ver **Figura 8**) y asignar esos segmentos a el grupo vacío como se muestra en la **Figura 9**. Este cambio ofrece un mejor agrupamiento que el k-means clásico para este problema.

1	2	3	4
200	540	0	700

Figura 6. Solución desbalanceada

4
700

Figura 7. Grupo con mayor número de elementos

3	4
350	350

Figura 8. Distribución de elementos al grupo con más miembros

1	2	3	4
200	540	350	350

Figura 9. Grupos Blanceados

A continuación, en la **Figura 10** se muestra el seudocódigo del algoritmo k-means. En las líneas 2 a la 4, se reciben los clusters procesados en la **sección 4.6.1.1** para ser normalizarlos, luego en la línea 5 al resultado de la normalización se le aplica el proceso para obtener los centroides de cada cluster. De la línea 6 a la 27 se realiza un bucle ***for*** e internamente en la línea 9 se calcula de los centroides el mejor grupo y en la 10 se actualiza el grupo actual.

Algoritmo K-means

```
1:  Function K-means (solucion)
2:      for i to numero_clusters do
3:          grupos_normalizados <- normalizar_grupos(cluster, solucion)
4:      end for
5:      centroides <- obtener_centroides(grupos_normalizados)
6:      for i to iteraciones do
7:          contador_cambios <-0
8:          for j to numero_segmentos do
9:              mejor_grupo <- calcular_mejor_grupo(centroides,j)
10:             grupo_actual <- solucion[j]
11:             if mejor_grupo != grupo_actual then
12:                 mejor_segmento<-obtener_segmento(grupos_normalizados[grupo_actual],j)
13:                 solucion[j] = actualizar_segmento(mejor_segmento,mejor_grupo,grupo_atual,j)
14:                 contador_cambios <- contador_cambios + 1;
15:                 if calcular_numero_grupos(solucion) != numero_clusters then
16:                     solucion <- balancear_resultado(solucion)
17:                     for k to numero_clusters do
18:                         grupos_normalizados <- normalizar_grupos(cluster,solucion)
19:                     end for
20:                 end if
21:                 centroides <-obtener_centroides(grupos_normalizados)
22:             end if
23:             if contador_cambios == 0 then
24:                 break
25:             end if
26:         end for
27:     end for
28:     return (solucion)
29: end Function
```

Figura 10. Pseudocódigo de K-means

De la línea 11 a la 22 se valida si el mejor grupo es diferente al grupo actual entonces se realiza el proceso de obtener un segmento de los grupos normalizados del cluster actual. De la línea 15 a la 20 se valida sí el número de grupos de la solución es diferente al número establecido de clusters, y si esto sucede, se realiza el proceso de balanceo explicado anteriormente. Una vez se valida se vuelve a realizar el proceso de calcular los centroides en la línea 21, Finalmente se retorna el resultado completo del proceso como se indica en la línea 20

4.6.1.3 RESTRICTED GROWTH STRINGS

En el problema se trabaja un número de clusters n en el rango de 2 a 10, eso implica que los 3.674 segmentos que hay en el dataset, deben quedar repartidos en “n” número de grupos y cada solución generada debe ser única, para lograr esto se usó una cadena de crecimiento restringido (Restricted Growth Strings, RGS), asegurando la unicidad de

cada solución, y controlando de una mejor forma las soluciones generadas en el K-means, evitando tener en cuenta soluciones previamente generadas.

Por ejemplo, se tiene una solución de la forma v = {a,b,c,d}, con una representación de 2 clusters así {1, 1, 2, 2 }, si no se usa una RGS puede aparecer otra solución con diferente representación, como el caso de {2, 2, 1, 1} que representa exactamente los mismos grupos, lo que implica que al ejecutar k-means y las meta-heurísticas se tendría en cuenta un espacio de representación con más permutaciones de las requeridas. La **Figura 11** muestra el algoritmo que toma un individuo representado con una cadena si restricción de crecimiento y retorna su equivalente único RGS.

Este algoritmo se basa en la solución propuesta en [48], a la cual se le han realizado pequeñas adaptaciones para poderlo usar en el contexto específico de la gestión de pavimentos. El cálculo de la cadena *RGS* consiste en generar un vector donde se cumpla con la restricción de crecimiento. Para este procedimiento se obtiene el mayor elemento de la cadena recibida, esto se lleva a cabo en la línea 2 y se asigna a una variable *K,* después se genera una matriz *swap* de dimensiones [k, 2] donde *K* es el número de filas y 2 es el número de columnas (línea 3). Esta matriz se utiliza para intercambiar los valores de la *RGS* actuales por los correctos. Por ejemplo, se tiene una *RGS* de diez elementos y cuatro grupos como se muestra en la **Figura 12**, la matriz de intercambio contiene los valores mostrados en la **Figura 13**, indicando el valor por el cual debe ser cambiado para generar la cadena con restricción de crecimiento. Tras el reprocesamiento se obtiene la *RGS* mostrada en la **Figura 14**.

Algoritmo Calcular RGS

```
1:  Function Calcular_RGS (individuo)
2:      k <- max(individuo)
3:      swap <- matrix(nrow = k, ncol = 2)
4:      for i = 1 to k do
5:          swap[i,1]=i
6:          swap[i,2]=-1
7:      end for
8:      tmp <- individuo[1]
9:      swap[tmp,2] <- 1
10:     individuo[1] <- 1
11:     maxt <- 1
12:     k <- 1
13:     for i = 2 to length(individuo) do
14:         if swap[individuo[i],2]!=-1 then
15:             individuo[i] <- swap[individuo[i],2]
16:         else
17:             maxt <- maxt+1
18:             swap[individuo[i],2] <- maxt
19:             individuo[i] <- maxt
20:         end if
21:         if k<individuo[i] then
22:             k <- individuo[i]
23:         end if
24:     end for
25:     return (individuo)
26: end Function
```

Figura 11. Pseudocódigo de RGS

3	1	2	2	4	1	1	1	3	3

Figura 12. Vector resultante del K-means

Para llegar a esa matriz, se asignan los índices de manera ascendente de uno hasta “n” (cantidad de clusters de la cadena) y los elementos de la segunda columna se llenan con el valor de -1 (líneas 4 a 7). Se almacena el primer valor de la cadena en una variable temporal (tmp), para tomar el valor almacenado de la matriz de intercambios en la posición [tmp, 2] como se ve en la **Figura 13**. Luego, se fija en uno la primera posición ya que, en R el valor base es uno, se crea una variable que contiene el mayor valor asignado (maxt) a la *RGS* hasta el momento (inicialmente uno) y se fija la variable k, con un valor en uno (líneas 8 a 12). Desde el segundo elemento de la cadena recibida, se comienza su recorrido y cuando se encuentre con un valor diferente a la bandera

asignada (-1) en la posición de la matriz swap [i,2] , se procede a hacer el intercambio (líneas 13 a 16). Si el valor en dicha posición no ha cambiado (-1), entonces se incrementa el mayor valor asignado (maxt) y en la posición [rgs[i],2] de la matriz tanto como en la posición "i" de la *RGS* se fija el valor del mayor valor asignado (maxt) (líneas 17 a 18). Después se verifica que el valor de *RGS* en la posición "i" sea mayor al valor asignando en k, en caso contrario se asigna a la variable de k el valor que contenga la cadena *RGS* en la posición "i" (líneas 21 a 23), este proceso se repite para el tamaño de la cadena recibida en la función. Después se retorna la cadena *RGS* la cual cumple con la restricción de crecimiento como se ve en la **Figura 14**.

1	2
2	3
3	1
4	4

Figura 13. Matriz de intercambios

1	2	3	3	4	2	2	2	1	1

Figura 14. RGS resultante

4.6.1.4 OBTENER FITNESS

Para obtener el fitness de una solución es necesario calcular el valor de fitness de cada grupo de esta. Para obtener los fitness de cada grupo, primero se toman los segmentos del grupo, es decir, los registros en su transformación logarítmica. Después se verifica que las variables de estos grupos tengan al menos dos valores distintos (varianza diferente de cero), las variables que no cumplan con esta condición se eliminan del conjunto de datos. Luego se ejecutan una serie de pasos que se explican detalladamente más adelante. Los pasos son los siguientes:

- Selección de variables (feature selection) usando la meta-heurística GBHS (***GBHS-FS***).
- Proceso del cálculo del ***valor P*** el cual es un subproceso interno de la selección de variables.
- Proceso de ***creación de la memoria armónica*** inicial para la selección de variables.
- Proceso de ***Improvisación*** que subsume las iteraciones de creación de nuevas soluciones usando las reglas de GBHS para la selección de variables.

GBHS FS

Como valor agregado a la propuesta de la investigación base de Khadka, buscando reducir el tiempo de ejecución y mejorar el proceso de selección de atributos, dicho proceso se realizó sin usar la combinatoria exhaustiva de variables propuesta por Khadka, sino utilizando la meta-heurística *Global-Best Harmony Search.* En este caso, la metaheurística recibe los datos (registros del dataset) pertenecientes al cluster en cuestión, además de un vector con todos los nombres de las variables explicativas continuas y categóricas del dataset y luego se procede a crear la memoria armónica como se explica en la línea 3 del pseudocódigo que se muestra en la **Figura 15**. Luego, de la línea 4 a la 7 se controla el máximo número de evaluaciones de la función objetivo que se pueden realizar y finalmente en la línea 8 se realiza el proceso correspondiente a las improvisaciones, donde se busca mejorar las variables seleccionadas para grupo/clúster en cuestión basado en el valor P.

Algoritmo Global Best Harmony Search para Feature Selection (GBHS-FS)

```
1:  Function GBHS Feature Selection (datos_transformados, vector_nombres)
2:      contador_efos <-0
3:      memoria_armonica <-crear_memoria_armonica (datos_transformados, vector_nombres,
        maximo_soluciones,contador_efos)
4:      validar_efos (contador_efos, memoria_armonica)
5:      if memoria_armonica == NULL then
6:          return NULL
7:      end if
8:      Combinaciones <- improvisaciones (memoria_armonica, datos_transformados, vector_nombres,
        contador_efos)
9:      return (combinaciones)
10: end Function
```

Figura 15. Mejor Búsqueda Armónica Global para selección de características

CALCULO DEL VALOR P

En el estado del arte cuando se obtiene el modelo de regresión lineal (en este caso sobre variables transformadas con logaritmo) de la selección de variables específica para un grupo/clúster, también se obtiene un resumen con la información de los errores típicos, el estadístico t y los valores p para cada uno de los coeficientes de las variables seleccionadas. De los valores p obtenidos se toma el máximo de ellos y lo comparan con el nivel de significancia que se quiere que tenga el modelo, por ejemplo, 0.005 si el valor p es menor se acepta como un modelo valido. En el presente trabajo de investigación se realizó un cambio que ayudó a mejorar el proceso de convergencia del valor BIC en los

algoritmos y consistió en tomar el valor p directamente de la función de R (*Lenguaje que se usa para realizar los experimentos)* que genera el modelo.

CREACIÓN MEMORIA ARMÓNICA

A pesar de que consiste en crear una población inicial, este proceso en GBHS FS es ligeramente distinto al previamente explicado. El anterior buscaba generar una solución de agrupamiento, este busca definir que variables en un grupo son las más importantes de acuerdo con el modelo de regresión no lineal que se busca establecer para cada grupo o cluster.

Para el proceso de la creación de la memoria armónica que se muestra en la **Figura 16**, es necesario como lo indica la línea 1, tener los datos de los segmentos transformados a logaritmo, también el vector de nombres que hace referencia a los nombres de las variables explicativas continuas y categóricas, además de un número máximo de soluciones a generar que permite controlar el número de evaluaciones de la función objetivo (EFOs) que se van a realizar en el algoritmo.

Lo importante en este proceso como se evidencia en la línea 3 a la 15, es que al crear los valores aleatorios se obtenga un modelo con su respectivo valor p (<= 0.05 o el nivel de significancia definido) y su valor de fitness sea aceptado como una buena solución. Si el valor no es aceptado con su nivel de significancia, se siguen creando modelos hasta que puedan ser aceptados, todo sin olvidar que el número de EFOs no se debe sobrepasar. En la línea 16 a la 22, si se obtiene una buena solución esta es agregada a la memoria armónica y se repite este proceso tantas veces como se haya definido el tamaño de la memoria armónica o hasta que el máximo de EFOs llegue a su límite. Finalmente, de la línea 23 a la 24 se ordena de menor a mayor respecto al valor del $R^2 ajustado$ y se envía el primer elemento de la memoria armónica que corresponde al mejor.

IMPROVISACIÓN

Continuando con los pasos mencionados en la sección anteriormente mencionada, se procede a explicar el proceso de generar las improvisaciones, el cual es un subproceso del GBHS FS al que se le realizan adaptaciones respecto al GBHS original con el fin de centrarlo en la selección o no de variables, es decir un problema binario. El pseudocódigo del algoritmo se presenta en la **Figura 17**. Se establecieron el número de iteraciones y se fijaron valores para el tono de la tasa de ajuste parMin, parMax y la tasa de

consideración de la memoria armónica (HMCR), un parámetro que controla la explotación/exploración.

Algoritmo Crear Memoria Armónica

```
1:  Function: crear memoria armonica(datos_transformados, vector_nombres, contador_efos, tamaño_memoria)
2:      for i to tamaño_memoria do
3:          repeat
4:              nombres_aleatorios <-valores_aleatorios(vector_nombres)
5:              contador_efos <- contador_efos + 1
6:              valor_P <- calcular_valor_P(vector_nombres, datos_transformados,nombres_aleatorios)
7:              if valor_[1] < nivel_significancia then
8:                  break;
9:              else
10:                 valor_P <- NULL
11:             end if
12:             if contador_efos == maximo_soluciones then
13:                 break
14:             end if
15:         end repeat
16:         if valor_P != NULL then
17:             memoria_armonica[i] <- crear_armonia(valor_P, nombres_aleatorios)
18:             validar_efos(contador_efos)
19:         else
20:             break
21:         end if
22:     end for
23:     memoria_armonica <- ordenar_memoria_armonica(memoria_armonica)
24:     return(memoria_armonica,contador_efos)
25: end Function
```

Figura 16. Pseudocódigo para crear memoria armónica de la selección de atributos

En la línea 3 se establece la tasa de ajuste del tono para cada improvisación, luego de la línea 4 a la 7 se obtiene una armonía a partir de la tasa de ajuste y la memoria armónica, a la cual se le aplica un proceso de validación. De la línea 8 a la 9 se hace la selección de los modelos de la armonía generada anteriormente y se le realiza a esos modelos el cálculo del valor p, el método de la línea 8 denomina *elegir nombres armonía* se explica a detalle más adelante y su pseudocódigo se evidencia en la **Figura 18**. De la línea 10 a la 16 se realiza la validación del valor p obtenido, donde se compara que este sea menor al nivel de significancia establecido, si es menor se genera una nueva armonía basada en el resultado anterior, se verifica que esta armonía no exista en la memoria armónica con el fin de poderla agregar a la memoria armónica, ordenar la memoria y eliminar la armonía más mala. Finalmente, en la línea 18 y 19 se genera el modelo final a partir del vector de nombres que llega como parámetro al algoritmo y la memoria armónica para luego retornar ese resultado.

Algoritmo Improvisaciones

```
1:  Function improvisaciones (memoria_armonica,datos_transformados, vector_nombres)
2:      for i to iteraciones do
3:          tasa_ajuste <- obtener_tasa_ajuste (parMax,parMin,iteraciones)
4:          for j to tamaño_armonia do
5:              nueva_armonia <- obtener_nueva_armonia(tasa_ajuste, memoria_armonica)
6:          end for
7:          validar_armonia(nueva_armonia)
8:          nombres_aleatorios <- elegir_nombres_armonia(nueva_armonia)
9:          valor_P <- calcular_valor_P(nombres_aleatorios, datos_transformados ,vector_nombres)
10:         if valor_P < nivel_significancia then
11:             nueva_armonia <- generar_nueva_armonia(nueva_armonia, nombres_aleatorios, valor_P)
12:             if existe_en_memoria_armonica(memoria_armonica, nueva_armonia) == FALSE then
13:                 memoria_armonica <- agregar_armonia_memoria_armonica(nueva_armonia)
14:                 memoria_armonica <- ordenar_memoria_armonica(memoria_armonica)
15:             end if
16:         end if
17:     end for
18:     combinacion_resultante <- asignacion_nombres_solucion(vector_nombres, memoria_armonica)
19:     return(combinacion_resultante)
20: end Function
```

Figura 17. Pseudocódigo improvisaciones

Como se menciona en la descripción del algoritmo de improvisaciones es necesario explicar el funcionamiento del método *elegir nombres armonía,* el cual es fundamental en el proceso (ver pseudocódigo en la **Figura 18**). En la línea 3 se inicializa un vector temporal y en la línea 4 se realiza el cálculo de la fórmula de la tasa de ajuste del tono, de la línea 5 a la 16 se realiza un bucle que va hasta el tamaño de las variables (parámetro que llega al inicio del algoritmo), internamente en el bucle para cada segmento de la solución existen tres diferentes casos, en la línea 6 se valida que al generar un número aleatorio entre 0 y 1, si su valor es menor al HMCR, se toma el segmento de la iteración actual de un valor aleatorio del tamaño de la memoria armónica como se indica en línea 7 y 8.

En las líneas 9 a 12 se genera un valor aleatorio entre 0 y 1 y se compara con el resultado del cálculo de la línea 4 del tono de la tasa de ajuste, si el resultado es menor se toma la mejor posición de la memoria armónica y se asigna en la posición actual del vector temporal inicializado de la línea 3, si el resultado no es menor se toma un valor aleatorio entre 0 y 1 y se asigna al vector temporal. De la línea 17 a la 20 se realiza el proceso de validar si en el vector temporal todos los elementos son iguales a cero, se asignan en un nuevo vector un valor aleatorio entre 0 y 1, después en el vector temporal se asigna el

valor de 1 a los índices de posición guardados en el vector posición. Finalmente, en la línea 22 se realiza el proceso de tomar del vector temporal todos los elementos que contienen uno en su valor y se asignan al vector de nombres, para luego retornar el vector de nombres que corresponde al nuevo modelo.

Algoritmo Elegir Nombres Armonía

```
1.  Function elegir nombres armonia (armonia, tamaño_variables)
2.      for i to iteraciones do
3.          vector_tmp
4.          par_v <- parMin + ((parMax-parMin) *(ite/iteraciones))
5.          for j to tamaño_variables do
6.              if valor_aleatorio(0,1) < HMCR then
7.                  solucion_temporal = memoria_armonica[valor_aleatorio(memoria_armonica.length)]
8.                  vector_tmp[j] = solucion_temporal[j]
9.                  if valor_aleatorio(0,1) < par_v then
10.                     solucion_temporal_2 = memoria_armonica[1]
11.                     vector_tmp[j] = solucion_temporal_2[j]
12.                 end if
13.             else
14.                 vector_tmp[j] = valor_aleatorio(0,1)
15.             end if
16.         end for
17.         if all(vector_tmp == 0)
18.             vector_posiciones = valor_aleatorio(1, tamaño_variables,2)
19.             vector_tmp[vector_posiciones] = 1
20.         end if
21.     end for
22.     vector_nombres = tranformar_nombres(vector_tmp)
23.     return(vector_nombres)
24. end Function
```

Figura 18. Pseudocódigo elegir nombres armonía

Nota: En la Linea 17 la funcion **all()** revisa que todas las posiciones del vector_tmp sean cero.

EJEMPLO DEL ALGORITMO IMPROVISACIONES

A continuación, se presenta un ejemplo del algoritmo improvisaciones donde se muestra en diferentes iteraciones el proceso.

Caso 1:

En la **Figura 19** se presenta la Memoria Armónica inicial (*Organizada de menor a mayor, siendo el menor, el mejor valor para el SSE*).

Posición	Armonías	SSE
1	1 0 0 … 1	-10
2	0 1 1 … 1	-8
3	1 0 1 … 0	-4
4	0 0 0 … 1	-1

Figura 19. Ejemplo de Memoria Armónica para caso 1

Valor aleatorio = 0.6
HMCR = 0.85 (tasa de consideración de la memoria armónica)
Iteración = 1
Tamaño de la memoria armónica = 4

Si se cumple la condición, de que el valor aleatorio es menor que la razón de exploración (HMCR) entonces se genera un número aleatorio entre uno y el tamaño de la memoria armónica. En este caso el ***valor aleatorio [1 – 4]*** fue 3 (ver primera celda de la **Figura 20**).

3		1	0	1	…	0	-4

Figura 20. Armonía en la posición 3, caso uno

De la armonía 3 se obtiene el valor que corresponda a la iteración actual, en este caso la posición uno de la armonía, como se aprecia resaltado en la **Figura 21**.

1	0	1	…	0

Figura 21. Armonía de la memoria armónica, caso uno

Ese valor, se asigna en la posición de la iteración actual en la nueva armonía que se pretende generar, como se muestra en la **Figura 22**.

1				

Figura 22. Nueva armonía, caso uno

Para el segundo caso se deben cumplir dos condiciones, la primera es que el primer valor aleatorio generado sea menor que la razón de exploración (HMCR), la segunda que un segundo valor aleatorio debe ser menor al valor dado por la ecuación (4) relacionado con la tasa de ajuste del tono (Pitch Adjustment Rate, PAR).

$$par_v = parMin + ((parMax - parMin) * (i/iteraciones))$$

Ecuación (4)

En este caso se toma el mejor individuo de la memoria armónica el cual corresponde a la posición número uno, dado que dicha memoria esta ordenada respecto al SSE de menor a mayor.

Caso 2:

La **Figura 23** muestra la Memoria Armónica inicial.

Posición	Armonías	SSE
1	1 0 0 … 1	-10
2	0 1 1 … 1	-8
3	1 0 1 … 0	-4
4	0 0 0 … 1	-1

Figura 23. Ejemplo de Memoria Armónica para caso dos

Valor aleatorio = 0.6
HMCR = 0.85 (Razón de exploración)
Iteración = 2
Tamaño de la memoria armónica = 4

Si el segundo valor generado aleatoriamente es menor que el resultado de la fórmula de la ecuación (4), entonces se toma la posición uno de la memoria armónica, como se aprecia en la **Figura 24**.

1		1	0	0	…	1	-10

Figura 24. Armonía en la posición 1, caso dos

De la memoria armónica se toma el valor de la iteración que en este caso es 2, que corresponde a la celda resaltada de la **Figura 25**.

1	**0**	0	…	1

Figura 25. Armonía de la memoria armónica, caso dos

Se asigna el valor de la memoria armónica en la nueva armonía, como se muestra en la **Figura 26**.

1	**0**				

Figura 26. Nueva Armonía, caso dos

El tercer caso se da cuando el caso *uno* no se cumple, es decir, cuando el primer número aleatorio es mayor que la razón de exploración (HMCR), entonces se asigna un valor aleatorio de 0 o 1 en la posición de la iteración actual, ejemplo:

Caso 3:

La **Figura 27** muestra la Memoria Armónica inicial.

Posición	Armonías	SSE
1	1 0 0 … 1	-10
2	0 1 1 … 1	-8
3	1 0 1 … 0	-4
4	0 0 0 … 1	-1

Figura 27. Ejemplo de Memoria Armónica para caso dos

Valor aleatorio = 0.9
HMCR = 0.85 (Razón de exploración)
Iteración = 3
Tamaño de la memoria armónica = 4

Dado que el valor aleatorio supera la razón de exploración, entonces se genera un número aleatorio de 0 o 1, y se asigna a la posición de la iteración actual. En este caso el ***Valor aleatorio [0-1]*** *de 1* se asigna en la celda resaltada en la **Figura 28**.

1	0	**1**			

Figura 28. Nueva Armonía caso tres

Este procedimiento se repite hasta llenar la armonía con el número de variables explicativas y categóricas que se tienen en el dataset (en este caso este valor es de 14). Una vez se obtiene la nueva armonía se valida que al menos contenga en dos posiciones el valor de 1, luego se sacan los nombres de las variables del vector resultante siendo 1 si la variable queda en la selección y 0 si se descarta. Luego se halla el valor p, con la selección resultante y el conjunto de segmentos en su forma logarítmica perteneciente a su respectivo cluster. Se verifica que el valor P, cumpla con el nivel de significancia establecido (el estado del arte lo define en 0.05), una vez se verifique, si el valor p cumple con el nivel de significancia aceptado, entonces se procede a verificar que la armonía generada anteriormente no se encuentre en la memoria armónica, si no existe ya, entonces la armonía se adiciona a la memoria armónica, se organiza la memoria armónica de menor a mayor basado en el SSE y se elimina la armonía con el mayor SSE. Este proceso se realiza para “n” iteraciones, en este caso 100 iteraciones, todo basado en el estado del arte. Una vez completado el proceso iterativo, se retorna la primera posición de la memoria armónica que es la mejor de todas las armonías.

4.6.1.5 CALCULAR BIC EN GENERAL

Para el cálculo del BIC en general, que es la función objetivo a minimizar (ver **sección 4.4**), se necesita tener el valor de los fitness de cada cluster de la solución dada, por ejemplo, si existen cinco clusters, deben existir cinco valores de fitness para calcular el BIC de esta solución. Este método se tomó del trabajo base y no fue necesario realizarle ningún ajuste.

4.7 ALGORITMOS

En el trabajo de investigación se proponen seis algoritmos, de las cuales tres son metaheurísticas de búsqueda global y tres son las variantes meméticas de estas. A continuación, se describe a detalle las adaptaciones necesarias aplicadas a cada una.

4.7.1 PARTICLE SWARM OPTIMIZATION (PSO)

Esta propuesta está inspirada en el algoritmo conocido en su traducción al español como, optimización por enjambre de partículas. El PSO original fue modificado en gran parte con el fin de adaptarlo al problema de CLR y los resultados prometedores en la experimentación animan a evaluarlo directamente como un algoritmo de clustering de datos.

Antes de ejecutar este algoritmo se debe realizar un preprocesamiento al dataset que ya fue explicado en la **sección 4.5**, además de la asignación de los parámetros del algoritmo. En la **Figura 29** se muestran los parámetros utilizados para la configuración del algoritmo.

Parámetros de entrada	
Inercia cognitiva inicial	*CP*
Inercia cognitiva final	*CPF*
Factor social inicial	*CG*
Factor social final	*CGF*
Momentum Inicial	*WI*
Momentum Final	*WF*
Tamaño de la población	Definido por el usuario
Máximas iteraciones	Definido por el usuario
Número de clusters	Definido por el problema
Nivel de significancia	Definido por el estado del arte

Figura 29. Parámetros de entrada PSO

4.7.1.1 INICIALIZACIÓN DE LA POBLACIÓN

Para inicializar el algoritmo es necesario conocer cómo se forma una población y como está compuesta internamente.

PARTICULA

El algoritmo de *PSO*, está compuesto por partículas, una partícula tiene las siguientes características: la posición actual, el valor de la función objetivo en este caso el BIC, la mejor posición encontrada en todos los puntos de búsqueda que ha visitado y su respetivo BIC. En el PSO propuesto no se usa el concepto de velocidad original del PSO continúo propuesto en la literatura. A continuación, en la **Figura 30** se muestra como es la estructura de una partícula. Además, en la **Figura 31** se muestra una población de partículas, población que está compuesta por “n” partículas donde el valor “n” es un parámetro del algoritmo.

Posición actual	BIC(*Posición actual*)	Mejor posición	BIC(*Mejor posición*)
3 3 6 .. 2	-100	2 6 6 .. 1	-200

Figura 30. Representación de una partícula

De la población se obtiene la mejor partícula global, esta es seleccionada basado en el mejor valor del BIC, esta partícula es fundamental para la meta-heurística ya que tiene como función guiar al resto de partículas hacia una mejor solución y se actualiza por cada iteración del algoritmo. En la **Figura 32** se muestra la representación de la mejor partícula global.

Posición	Partícula			
1	Posición actual	BIC	Mejor posición	BIC
	3 3 6 .. 2	-100	2 6 6 .. 1	-200
2	Posición actual	BIC	Mejor posición	BIC
	1 2 6 .. 1	-54	2 6 6 .. 2	-95
3	Posición actual	BIC	Mejor posición	BIC
	5 3 3 .. 4	-300	2 5 3 .. 6	-400
4	Posición actual	BIC	Mejor posición	BIC
	5 5 5 .. 5	-10	2 6 6 .. 5	-50

Figura 31. Representación de una población PSO

Posición actual	BIC
5 3 3 .. 4	-300

Figura 32. Representación de la mejor partícula global

El algoritmo PSO propuesto, se presenta en la **Figura 33**. En la línea número dos, el método "*inicializar variables"* hace referencia a los parámetros establecidos en la literatura y previamente presentados en la **Figura 29**. En la línea 3 se inicializa la población como se explicó anteriormente en la **sección 4.6**, luego se asigna la población y el cálculo de los individuos creados en la inicialización. Las líneas 4 y 5 indican que se realiza un proceso de validación, donde primero se calcula al mejor individuo y posteriormente se revisa que si al inicializar la población se crean el número límite fijado para los individuos, se salga del algoritmo.

En la línea 6 se entra a un ciclo *que* va hasta el número de iteraciones máximas (EFOs). En la línea 7 se establecen unas fórmulas (que se presentan a continuación y se explican en detalle más adelante) para la definición de las nuevas posiciones de las partículas y que están adaptadas al problema específico del proyecto [49].

Fórmula para el Momentum

$$wt = wi - (wi - wf) * \left(\frac{iteración}{MaxIteraciones}\right)$$

Fórmula para la inercia cognitiva

$$cpt = cp - (cp - cpf) * \left(\frac{iteración}{MaxIteraciones}\right)$$

Fórmula para el factor social

$$cgt = cg + (cgf - cg) * \left(\frac{iteración}{MaxIteraciones}\right)$$

Algoritmo *Particle Swarm Optimization*

```
1:  Function Algoritmo PSO
2:      Inicializar_variables()
3:      poblacion <- inicializar_poblacion_global(tamaño_poblacion, contador_individuos)
4:      individuo_mejor_global <- mejor_individuo(poblacion)
5:      limite_efos(contador_individuos)
6:      for i to iteraciones do
7:          probabilidad <- calculo_probabilidad(i, iteraciones, wi,wf,cp,cpf,cg,cgf)
8:          for j to tamaño_poblacion do
9:              particula_generada <- obtener_nueva_particula(probabilidad)
10:             validar_nueva_particula(particula_generada)
11:             particula_generada <- calcular_rgs (particula_generada)
12:             fitness <- obtener_fitness_grupos(particula_generada)
13:             validar_fitness(fitness)
14:             limite_efos(contador_individuos)
15:             BIC <- calculo_general_fitness(fitness)
16:             actualizar_mejor_local(BIC)
17:             limite_efos(contador_individuos)
18:         end for
19:         actualizar_mejor_global(poblacion)
20:         hill_climbing(individuo_mejor_global)
21:     end for
22: end Function
```

Nota: La línea 20 solo se ejecuta en la presentación memética del algoritmo PSO.

Figura 33. Optimización por Enjambre de Partículas para CLR

En la línea 8 se ingresa a un ciclo que va hasta el tamaño de la población, el cual se fija al inicio del algoritmo. En la línea 9 se crea la nueva partícula de acuerdo con el vector probabilidad, que define el impacto de la posición de la mejor solución global del enjambre, el impacto de la posición del mejor histórico de la partícula y el impacto de la posición actual en la nueva posición de la partícula.

En la línea 10 en el proceso de *"validar la nueva partícula"*, garantiza que la nueva partícula sea diferente a la que salió del proceso de mutación anterior *"obtener nueva partícula"*, si no es así se realiza una pequeña mutación de algunos individuos. Luego se compara si el número de clusters de la población es diferente al número de clusters establecidos al inicio, si es así se realiza un proceso de balanceo el cual tiene como prioridad evitar que se desaparezcan los grupos. En la línea 11 se hace el proceso de RGS explicado en la **sección 4.6.1.3**. De la línea 12 se obtienen los modelos de la población de cada grupo y su fitness como se explicó en la **sección 4.6.1.4**.

Luego en la línea 13 se realiza la validación de la solución. Si esta no es válida el proceso se repite hasta encontrar una que si lo sea. El límite de parada de este ciclo interno es que el número del contador de individuos llegue hasta su límite. En la línea 14, se valida en número de evaluaciones de la función objetivo en caso de ser el límite del número de individuos se detiene el proceso y continua con la siguiente semilla o finaliza la ejecución dependiendo el caso.

En la línea 15 y 16, se realiza el cálculo de todos los modelos ya generados con el BIC y se revisa cuál fue la mejor solución y se actualiza dependiendo si la mejor solución supera la local. Finalmente se aumenta el número de individuos creados y en la línea 17 se valida que estos no superen el límite, de lo contrario se finaliza la ejecución y se continúa con la siguiente semilla o se termina el proceso. Por último, en la línea 19 se valida cuál es el individuo mejor global con el ya existente y se comparan, de ser mejor se actualiza.

4.7.1.2 DEFINICIÓN DE LA NUEVA POSICIÓN DE UNA PARTICULA

La línea 9 de la **Figura 33** encierra el principal cambio hecho al PSO original. Dicho cambio se basa en la forma como se seleccionan las nuevas posiciones de una partícula basada en su posición actual, su historia (memoria cognitiva) y la mejor solución del enjambre (memoria social). La **Figura 34** muestra como las fórmulas del cálculo del

momentum, inercia cognitiva y factor social afectan la asignación de la nueva posición en una partícula.

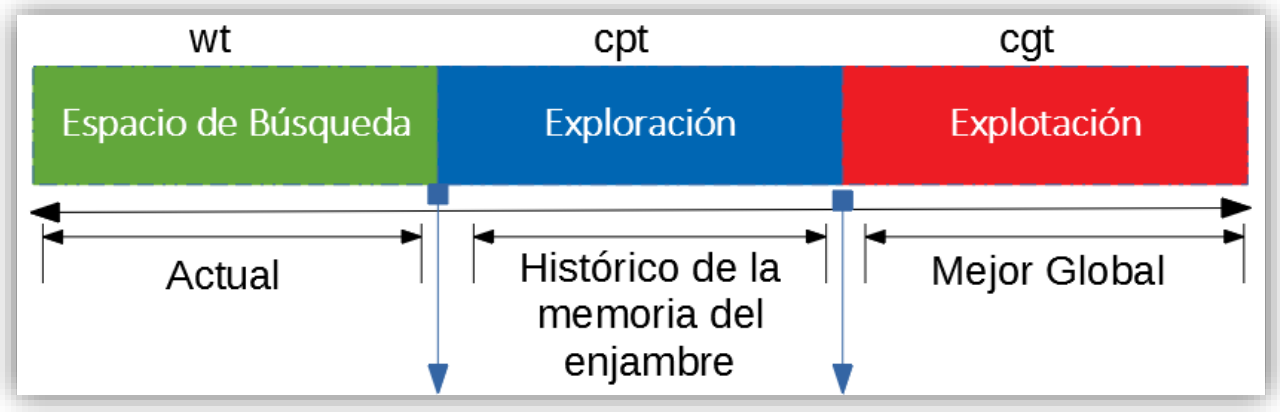


Figura 34. Gráfica de los rangos entre las probabilidades en el PSO

Se tiene una probabilidad *momentum (wt),* una probabilidad *de la inercia cognitiva (cpt)* y una probabilidad de *factor social (cgt),* con las cuales se definirá la nueva posición de la partícula. Todo inicia con la generación de un número aleatorio entre 0 y 1, si este número es menor que el momentum (wt), significa que el mejor individuo debe sacarse de la posición actual, pero si el número aleatorio, cae en el rango de ser mejor a la suma de (*wt + cpt)*, se saca del histórico mejor visitado (de la inercia cognitiva) en este caso como se indica en la **Figura 34**, es un espacio de exploración, pero si por el contrario el valor es mayor, significa que está en el rango del factor social (cgt) y la mejor solución se saca del mejor del mejor global, lo que implica hacer énfasis en la explotación de la mejor solución encontrada por el enjambre.

En la formulación original de *PSO* se maneja el concepto de velocidad, pero en este problema uno de los cambios significativos es eliminar ese concepto. La velocidad se usa para cambiar la dirección y la magnitud de la búsqueda, pero en el caso de CLR desvía el camino y hace que PSO se comporte como una búsqueda aleatoria.

Los valores de los parámetros de momentum, inercia cognitiva y factor social no son estáticos, ellos cambian conforme avanza el número de iteraciones conforme se muestra en la **Figura 35**. El valor de probabilidad cognitiva *(cp)* inicia en 2.5 y decrece linealmente, pero el factor social inicia en 0.5 y va aumentando linealmente con el paso de las iteraciones, esto hace que al principio se privilegie la experiencia de cada partícula pero con el paso de las iteraciones el algoritmo haga más explotación alrededor de la mejor

solución del enjambre. El componente de momentum que inicia en 0.9 y termina en 0.4 con un decremento también lineal, aumenta la probabilidad de seleccionar valores que giran alrededor de la partícula pero con el paso de las iteraciones ese peso se va perdiendo.

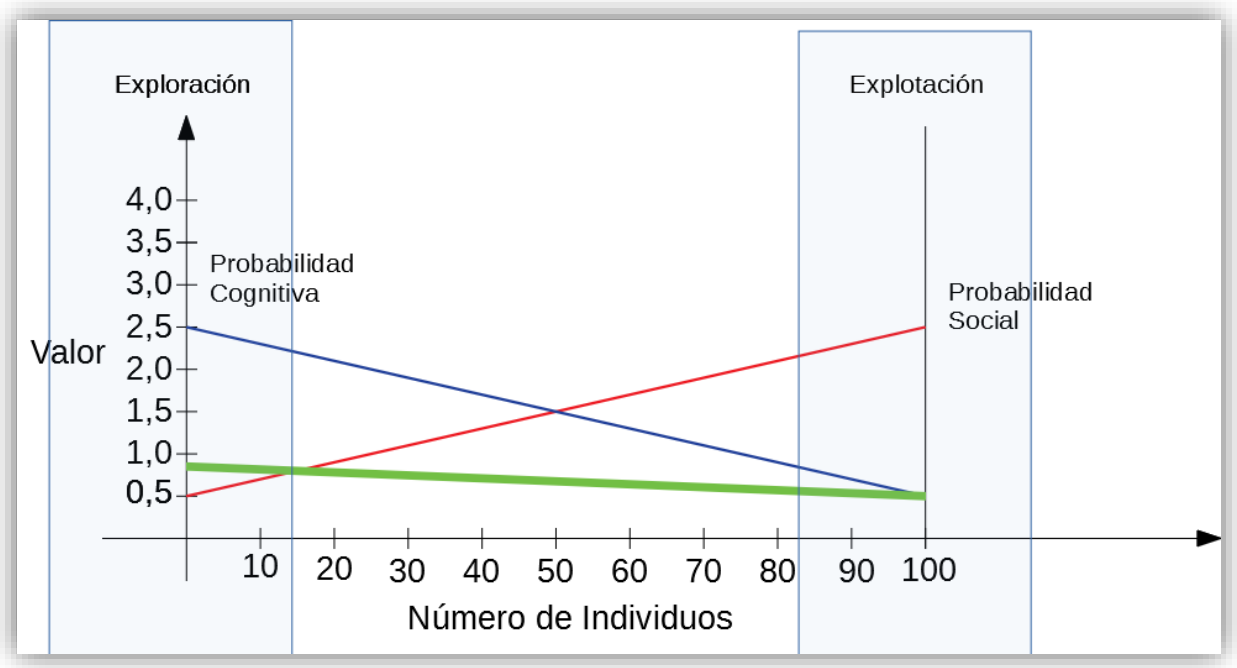


Figura 35. Esquema en el plano del comportamiento de las probabilidades

4.7.1.3 PSO MEMÉTICO

La variante memética del algoritmo *Particle Swarm Optimization,* se realizó integrando al final de la actualización de las posiciones de las partículas en la población, al algoritmo de búsqueda local, Hill Climbing. Proceso con el cual se busca realizar una explotación de los mejores individuos globales de cada iteración. Para ejecutar el PSO en su forma memética simplemente se activa la línea 20 y con ello se llama al pseudocódigo de la **Figura 36**.

Se recibe una solución candidata, que es la solución actual de la iteración del algoritmo que está en ejecución, en la línea 2 se establece un ciclo que va de uno hasta el máximo de iteraciones internas para el algoritmo de ascenso a la colina, luego en la línea 3 se realiza una copia de la solución con la cual se va a trabajar en los siguientes procesos y se realiza el *tweak (*proceso explicado más adelante en la **Figura 37**), una vez asignado

el resultado del tweak, en la línea 4 se valida la condición que si el BIC de la solución generada en el tweak es menor que la solución candidata actual, se actualice la solución como se indica en la línea 5. Una vez se cumplan las máximas iteraciones se sale del ciclo y finalmente se retorna la mejor solución encontrada como se indica en la línea 8.

Algoritmo *Hill Climbing*

```
1:  Function Hill Climbing (solucion)
2:      for i to max_iteraciones do
3:          R <- Tweak (copia(solucion))
4:          if R.BIC < solucion.BIC then
5:              solucion <- R
6:          end if
7:      end for
8:      return (solucion)
9:  end function
```

Figura 36. Pseudocódigo de Ascenso de Colina

Para el proceso del Tweak son necesarios lo siguientes parámetros: copia de la solución que se va a trabajar y el tamaño de la mutación. En la línea 2 del algoritmo se crea un ciclo que va hasta la línea 9, donde se repite este proceso hasta que la solución generada sea válida. Internamente en este bucle de la línea 3 a la 5 se realiza una mutación a la copia de la solución, esta mutación genera un valor aleatorio entre el rango de clusters aceptados y se asigna en una posición aleatoria de la solución, este proceso se realiza hasta el tamaño definido para la mutación como se indica en la 3. Luego se realiza el proceso de RGS explicado en la **sección 4.6.1.3**. En la línea 7 se realiza el proceso de obtener fitness de los grupos, donde separa cada uno de los grupos y hace una selección de caracteres como se explica en la **sección 4.6.1.4**. En la línea 8 se verifica que el valor de fitness de los grupos no sea nulo, si se sale del ciclo significa que la solución es válida si no se retorna el valor nulo, en la línea 10 se realiza el cálculo de la función objetivo BIC para todos los modelos y en la línea 11 se crea un nuevo individuo con los nuevos parámetros de posición y BIC.

Algoritmo Tweak

```
1:  Function Tweak(copia, tamaño_mutacion)
2:      repeat
3:          for i to tamaño_mutacion do
4:              mutar_solucion(copia,valor_aleatorio)
5:          end for
6:          posicion <- calcular_rgs(copia)
7:          fitness <-obtener_fitness_grupos(posicion)
8:          validar_fitness(fitness)
9:      end repeat
10:     BIC <- calculo_general_fitness(fitness)
11:     individuo <- crear_nuevo_individuo(posicion,BIC)
12:     return (individuo)
13: end Function
```

Figura 37. Pseudocódigo Tweak

4.7.2 GBHS

Similar a PSO, se debe hacer el preprocesamiento del dataset conforme se presentó en la **sección 4.5.** En la **sección 4.6.1.4,** se explica el funcionamiento de un GBHS FS interno para la selección de variables (feature selection), ya como algoritmo principal tiene unas pequeñas variaciones las cuales se presentan en el pseudocódigo. Es necesario tener en cuenta que en este proceso se considera el valor del BIC para evaluar la calidad de la solución de cluster completa y no el valor del SSE como se hace en GBHS-FS para cada grupo específico de la solución completa. En la **Figura 38** se muestran los parámetros de entrada del algoritmo y el seudocódigo en la **Figura 39**.

En la línea 2 del algoritmo GBHS, se inicializan las variables, proceso que tiene que ver con el ajuste de todos los parámetros de entrada mostrados en la **Figura 38**. Luego en la línea 3, se inicializa la población global, proceso explicado anteriormente en la **sección 4.6,** teniendo en cuenta que el individuo para este algoritmo se llama armonía y va a tener su respectivo BIC que para el caso es el valor de fitness. En la línea 4, se valida el límite de evaluaciones de la función objetivo, teniendo en cuenta el contador de individuos como criterio de parada del algoritmo. En la línea 5, se procede a ordenar la memoria armónica de menor a mayor para obtener el mejor individuo de la memoria, donde la primera posición indica que es el mejor, en caso contrario se entra a la función de improvisaciones principales en la línea 7. A continuación se muestra el pseudocódigo de improvisaciones principales en la **Figura 40**.

Parámetros de entrada	
Tamaño de la memoria armónica	HMS
Razón de Exploración	HMCR
Tono de ajuste mínimo	parMin
Tono de ajuste máximo	parMax
Número de iteraciones	Establecido por el usuario
Contador de Individuos	Establecido por el usuario
Tamaño de la población	Establecido por el usuario

Figura 38. Parámetros de entrada para GBHS

Algoritmo *Global Best Harmony Search*

```
1:  Function Global Best Harmony Search
2:      inicializar_variables()
3:      inicializar_poblacion_global(tamaño_poblacion,contador_individuos)//poner que devuelve
4:      if limite_efos(contador_individuos)== TRUE then
5:              mejor_armonia_global <- ordenar_memoria_armonica(memoria_armonica)
6:      else
7:              mejor_armonia_global <- improvisaciones_principales(memoria_armonica,contador_individuos)
8:      end If
9:  end Function
```

Figura 39. Pseudocódigo de La Mejor Búsqueda Global Armónica

El proceso de “Improvisaciones Principales” es similar al ya explicado en la **sección 4.6.1.4**. La diferencia está en que las armonías ya no contienen valores binarios si no valores que van desde uno hasta “k” grupos. Como se muestra en la **Figura 41** un ejemplo de una población aleatoria.

Otra de las diferencias relevantes en este proceso es que se realiza un balanceo de grupos para evitar que desaparezcan. Una vez balanceados los resultados, se aplica el proceso de RGS explicado en la **sección 4.6.1.3**, luego con el resultado, se crea una nueva armonía y se valida si se acepta o no como solución, en caso de que no sea válida, se crea una nueva solución respetando el criterio de parada. Luego, se procede a comprobar que la nueva armonía no exista en la memoria armónica y si no existe se agrega a la memoria, luego se ordena de menor a mayor y se elimina la última posición que corresponde a la peor solución de la memoria armónica, finalmente, cuando se terminen las iteraciones se retorna la mejor solución.

Algoritmo *Improvisaciones Principales*

```
1:  Function improvisaciones_principales (memoria_armonica, contador_individuos)
2:      for i to iteraciones do
3:          tasa_ajuste <- obtener_tasa_ajuste (parMax,parMin,iteraciones)
4:          for j to tamaño_armonia do
5:              nueva_armonia <- obtener_nueva_armonia(tasa_ajuste, memoria_armonica)
6:          end for
7:      end for
8:      validar_nueva_armonia(nueva_armonia)
9:      nueva_armonia <- calcular_rgs(nueva_armonia)
10:     fitness <- obtener_fitness_armonia(nueva_armonia)
11:     validar_fitness(fitness)
12:     limite_efos(contador_individuos)
13:     memoria_armonica <- existe_en_memoria_armonica(nueva_armonia, memoria_armonica)
14:     Hill_climbing(memoria_armonica)
15:     return (memoria_armonica[1])
16: end Function
```

Nota: Para ejecutar el algoritmo en su modo memético se agrega la línea 14, en caso contrario se omite.

Figura 40. Pseudocódigo improvisaciones principales

Posición	Armonías	BIC
1	1 3 5 … 4	-100
2	2 1 6 … 2	-80
3	3 2 1 … 5	-40
4	4 5 2 … 3	-10

Figura 41. Representación de la memoria armónica

4.7.2.1 GBHS MEMÉTICO

Para el proceso memético se incluye el Hill Climbing con el fin de realizar al final de cada iteración más explotación para la mejor solución. Este proceso memético está explicado en la **sección 4.7.1.3**.Para ejecutar el modo memético de GBHS se debe activar la línea 14 del pseudocódigo.

4.7.3 EVOLUCIÓN DIFERENCIAL

En este algoritmo meta-heurístico, también es necesario realizar el preprocesamiento explicado previamente en la **sección 4.5**, realizado también en los algoritmos *PSO* y *GBHS*. La **Figura 42** muestra el seudocódigo del algoritmo de evolución diferencial.

Algoritmo Differential Evolution

```
1:  Function Differential Evolution
2:      poblacion <- inicializar_poblacion_global(tamaño_poblacion, contador_individuos)
3:      limite_efos(contador_individuos)
4:      individuo_mejor_global <- mejor_individuo(poblacion)
5:      for i to iteraciones do
6:          poblacion <- generacion(poblacion, individuo_mejor_global , contador_individuos)
7:          if limite_efos (contador_individuos) == TRUE then
8:              actualizar_mejor_global(individuo_mejor_global, poblacion)
9:          end if
10:         if poblacion[i] < individuo_mejor_global then
11:             actualizar_mejor_global(individuo_mejor_global, poblacion)
12:         end if
13:         Hill_climbing(mejor_individuo)
14:     end for
15:     return(individuo_mejor_global)
16: end Function
```

Nota: Para ejecutar el algoritmo en modo memético se agrega la línea 13, de lo contrario se omite

Figura 42. Pseudocódigo Evolución Diferencial

En la línea 2, se realiza el proceso de "inicializar población global" explicada en la **sección 4.6**, luego en la línea 3 se realiza el proceso de validar el número de evaluaciones a la función objetivo, teniendo en cuenta el contador de individuos como criterio de parada del algoritmo. En la línea 4 se realiza el proceso de calcular el mejor individuo con el fin de organizar de menor a mayor los individuos basados en su BIC, donde cada generación va a retornar la primera posición de la memoria ya que contiene el mejor BIC. En la línea 5 se entra a un ciclo que va desde uno hasta el número de iteraciones, luego en la línea 6 se realiza la función de generación que se explica a continuación.

Un individuo en ED está compuesto por una posición y su correspondiente BIC tal como se muestra a continuación en la **Figura 43**.

Posición					BIC
2	3	3	…	5	-100

Figura 43. Representación de un individuo ED

Una población en ED es un conjunto de individuos como se muestra en la **Figura 44**.

Posición	Individuos					
1	Posición					BIC
	2	3	3	…	5	-100
2	Posición					BIC
	1	4	3	…	1	-90
3	Posición					BIC
	5	3	5	…	3	-70
4	Posición					BIC
	2	6	5	…	2	-110

Figura 44. Representación de una población en ED

En la Generación de un nuevo de individuo se realizan los siguientes pasos: primero se selecciona el padre, este proceso se realiza con una selección al azar de un individuo de la población a excepción del valor actual de la iteración que fue mencionado anteriormente.

Como ejemplo el valor aleatorio es **3** *(Padre)* como se muestra en la **Figura 45**.

3	Posición					BIC
	5	3	5	…	3	-70

Figura 45. Representación de la selección de un padre en ED

Ya con el padre seleccionado se necesita de otro individuo para poder realizar la perturbación, este individuo será el mejor global de la población, como ya se tienen los dos individuos necesarios, se procede a realizar la perturbación, la cual consiste en mutar a los individuos anteriores, eso viene dado de la siguiente manera: Se establece un valor de cruce (*F*), por ejemplo 0.8, luego se genera un valor aleatorio entre 0 y 1, para el ejemplo de 0.5, si el valor es menor al F, entonces el elemento i-ésimo (para el ejemplo el valor de i = 2 ya que se encuentra en la segunda iteración del ciclo) del mejor individuo en este caso es el individuo número 4 (ver **Figura 46**), ya que contiene el valor más bajo de BIC de la población.

	Posición					BIC
4	2	6	5	…	2	-110

Figura 46. Representación del mejor individuo seleccionado

Es asignado el elemento i-ésimo del individuo seleccionado conforme a la **Figura 47.**

	Posición					BIC
3	5	6	5	…	3	-70

Figura 47. Representación del padre seleccionado

Esto pasos se realizan para todos los elementos del individuo.

Después de completar el proceso de perturbación, se retorna el nuevo individuo el cual es enviado a un proceso de cruce con el individuo i-ésimo de la población. El cruce se define de la siguiente manera: se generan dos números aleatorios uno que varía entre 0 y 1, el otro valor podría tener un rango de uno al número de segmentos 3674, entonces se genera la condición donde si el valor es mejor que la taza de cruce que es por ejemplo 0.3 o el elemento i-ésimo sea igual al segundo valor aleatorio, en ese caso se deja el valor del individuo actual en el nuevo individuo cruzado, si no se cumple la condición entonces el valor i-ésimo del vector permutado se asigna al nuevo individuo.

Una vez realizado el proceso de generación en la línea 7 a la 9 se valida el número de evaluaciones a la función objetivo teniendo en cuenta como criterio de parada el contador de individuos, en la línea 10 se valida que si el valor de fitness del individuo creado es mejor que la global se actualiza el mejor global. Cuando se terminen de ejecutar todas las iteraciones o se detenga el algoritmo por su criterio de parada, se retorna el mejor resultado de todo el proceso.

4.7.3.1 EVOLUCIÓN DIFERENCIAL MEMÉTICO

Al igual que en los anteriores algoritmos PSO y GBHS se incluye una variante memética incluyendo Hill Climbing. Para ejecutar el modo memético de este algoritmo es necesario activarla la línea 13 del pseudocódigo correspondiente en la **Figura 33**.

CAPÍTULO 4

5 EXPERIMENTACIÓN

En este apartado, se presentan los resultados de los algoritmos meta-heurísticos PSO, GBHS y ED y su correspondiente version memetica (hibridado con HC) comparandose entre si y frente a los resultados del estado del arte (Recocido Simulado). Además del dataset utilizado para los experimentos y el ajuste de los parámetros para los algoritmos propuestos.

El proceso de experimentación fue realizado en un servidor remoto proporcionado por la Universidad de Nevada, Las Vegas, Estados Unidos, el cual cuenta con las siguientes características:

Software:

- Windows Server 2008 R2 con la arquitectura de 64 btis.

Hardware:

- Procesador Intel Xeon CPU X7560 2.26 Ghz de 64 núcleos
- Memoria Ram 256 Gb tipo DDR3
- Disco duro de 5.5 Terabytes

A continuación se presenta la lista del software necesario para la ejecución de los experimentos

Programas:

- R (Lenguaje)
- R-studio. (IDE)

Nota importante: Se debe tener en cuenta que al replicar la experimentación los resultados pueden variar un poco sino se utilizan las mismas semillas o si se usa una arquitectura de procesador diferente al que se usó en el presente trabajo.

5.1 DESCRIPCIÓN DEL DATASET

El dataset utilizado en el proceso de experimentación, como se menciona en la **sección 4.3** y la **sección 4.3.1**, explica a detalle el contenido de este, además de la función de cada variable en el problema. Este dataset fue utilizado en la investigación que actualmente es el estado del arte.

5.2 ALGORITMO DEL ESTADO DEL ARTE

El algoritmo reportado en el estado del arte actual es el Recocido Simulado, meta-heurística con la cual realizaron el proceso para estimar los modelos de gestión de pavimentos por clusterwise con regresiones lineales y no lineales. Determinaron como 16 el número máximo de clusters factibles para los datos utilizados. El algoritmo de solución exploró la cantidad posible de clústeres, es decir, K = 2 hasta 16, para buscar la cantidad óptima de clusters. Inicialmente, BIC fue de -26.955; y después de 1.437 iteraciones, el algoritmo encontró la solución óptima con BIC de -30.010.

5.3 CONFIGURACIÓN DE LOS PARÁMETROS

A continuación se muestra la configuración de los parámetros utilizados en cada algoritmo, estos valores están basados en la literatura, además se configura el software que contiene los scritps con el fin de que haga de manera automática el proceso iterativo de generar 30 experimentos con diferente semilla como lo aconseja la literatura para que al finalizar cada algoritmo se obtenga por cada cluster la mejor solución respecto a la función objetivo.

5.3.1 PARÁMETROS PARA EL PSO

La siguiente **Figura 48** muestra los valores que se utiliza para configurar de manera correcta el algoritmo de optimización por enjambre de partículas como se explica en la **sección 4.7.1.**

Parámetros de entrada	
Inercia cognitiva inicial	2.5
Inercia cognitiva final	0.5
Factor social inicial	0.5
Factor social final	2.5
Momentum Inicial	0.9
Momentum Final	0.4
Tamaño de la población	5
Máximas iteraciones	200
Numero de clusters	10
Nivel de significancia	0.05
Número de individuos	100
Número de EFOS	100

Figura 48. Valores de los parámetros de entrada para el algoritmo PSO

Para la versión memética del algoritmo PSO, se utilizan los parámetros de la figura anterior además de la configuración del algoritmo ascenso de colina con 5 iteraciones, en cada iteración se muta la solución en un 2,1% (La mutación representa mover 80 segmentos de forma aleatoria en la solución).

5.3.2 PARÁMETROS PARA EL GBHS

La **Figura 49** contiene los valores que se usan en la configuración del algoritmo de la mejor búsqueda global armónica, como se explica en la **sección 4.7.2**.

Parámetros de entrada	
Tamaño de la memoria armónica	5
Razón de Exploración	0.85
Tono de ajuste mínimo	0.1
Tono de ajuste máximo	0.4
Número de iteraciones	200
Número de individuos	100
Número de EFOS	100
Nivel de significancia	0.05

Figura 49. Valores de los Parámetros de entrada del algoritmo GBHS

Para la versión memética del algoritmo GBHS, se utilizan los parámetros de la tabla anterior, además de la configuración del algoritmo ascenso de colina conformo se explica en la sección previa de PSO.

5.3.3 PARÁMETROS PARA ED

La **Figura 50** muestra los valores que se utiliza para configurar de manera correcta el algoritmo Evolución Diferencial como se explica en la **sección 4.7.3.**

Parámetros de entrada	
Probabilidad de mutación	0.8
Tasa de cruce	0.3
Tamaño de la población	5
Número de iteraciones	200
Número de individuos	100
Número de EFOS	100
Nivel de significancia	0.05

Figura 50. Valores de los parámetros de entrada del algoritmo ED

Para la versión memética del algoritmo ED, se utilizan los parámetros de la figura anterior y para el algoritmo de ascenso de colina, la configuración mencionada en el **apartado 5.3.1**.

5.4 RESULTADOS OBTENIDOS Y COMPARACIÓN

En el proceso de experimentación, se logró optimizar el tiempo de los resultados utilizando el servidor remoto proporcionado por la Universidad de Nevada, las Vegas, en el cual se pueden correr en paralelo varias sesiones de R-studio a la vez, por ello, se lanzaron 30 sesiones de R-studio en paralelo por meta-heurística, en cada sesión se configura el algoritmo para que realice el proceso de 10 semillas, con el fin de usar 3 sesiones por cluster los cuales tardaban un tiempo aproximado de 5 días para terminar sus 30 ejecuciones.

El proceso anterior se hizo con clusters desde k = 2 hasta k = 10, a continuación, se muestran los resultados obtenidos de los algoritmos propuestos en el presente trabajo de investigación de las meta-heurísticas PSO, GBHS, ED y sus versiones meméticas representados en la **Figura 51.**

5.4.1 COMPORTAMIENTO DE LOS ALGORITMOS

Los datos que se muestran a continuación en la **Figura 51,** corresponden a los resultados obtenidos del comportamiento promedio de los algoritmos en su mejor valor BIC de las 30 semillas por cada número de agrupaciones.

Cluster	PSO		GBHS		ED	
	Normal	Memético	Normal	Memético	Normal	Memético
K = 2	**-46257,40682**	-46228,5649	-29507,55999	-29509,45297	-29526,89525	-29499,93361
K = 3	**-46722,49749**	-46708,46144	-29993,23992	-29987,85497	-29992,94547	-29982,27971
K = 4	**-46993,91395**	-46963,7049	-30976,30866	-30988,73754	-30981,3805	-30961,50538
K = 5	**-47458,11252**	-47438,61575	-31852,92408	-31853,97075	-31803,8751	-31784,75093
K = 6	**-48090,83111**	-48057,44805	-32149,98688	-32084,60984	-31997,02466	-31981,76212
K = 7	**-48357,77031**	-48333,83123	-32436,46404	-32383,62616	-32341,04887	-32324,91584
K = 8	**-48458,10224**	-48432,61748	-32683,33016	-32657,81139	-32584,80033	-32580,69162
K = 9	**-48693,31388**	-48670,72722	-32859,82402	-32831,95333	-32759,51957	-32756,89835
K = 10	**-48928,39363**	-48894,3689	-33018,17322	-33008,87397	-32938,68933	-32928,97524

Figura 51. Comportamiento de los algoritmos

Con base en los resultados de la **Figura 51,** es posible realizar un análisis estadístico en el cual se determina un ranking y la dominancia que tiene cada algoritmo a través del test de Friedman y de Wilcoxon.

5.4.2 ANALISIS ESTADISTICO

Los datos evidenciados en la **Figura 53**, se utilizaron para realizar el test de Friedman y de Wilcoxon (resultados evidenciados en anexos). Con el test de Friedman se puede evidenciar que las meta-heurísticas usadas en este trabajo son mejores que el estado del arte ya que sus resultados son estadísticamente significativos debido a que los resultados de la prueba son menores a 0.005, lo que genera un ranking de cual algoritmo es mejor en el análisis (ver **Figura 52**).

Algoritmo	Ranking
PSO Normal	1
PSO Memético	2
GBHS Normal	3.5556
GBHS Memético	3.8889
ED Normal	4.5556
ED Memético	6
SA	7

Figura 52. Resultados de la prueba de Friedman

Con el fin de determinar la dominancia del algoritmo que sobresale de los utilizados en la experimentación. Los resultados arrojados por el test de Wilcoxon se muestran en la **Figura 53.** En esta figura el cuadro negro (■) indica que el algoritmo de la fila domina al algoritmo de la columna, el punto blanco (o) indica que el algoritmo de la columna domina al de la fila y la casilla vacía indica que no es posible establecer una dominancia de un algoritmo sobre el otro. Además, los resultados por debajo de la diagonal tienen un nivel de significancia del 0.95 y los que están por encima de 0.90.

Algoritmo	(1)	(2)	(3)	(4)	(5)	(6)	(7)
PSO Normal (1)	-		■	■	■	■	■
PSO Memético (2)		-				■	■
GBHS Normal (3)			-				■
GBHS Memético (4)	o			-			■
ED Normal (5)	o				-		
ED Memético (6)	o					-	
SA (7)	o	o	o	o			-

Figura 53. Resultados de la prueba de Wilcoxon

Es claro que el algoritmo que domina la mayoría de los algoritmos es el PSO normal, además la mayoría de los algoritmos propuestos en el presente trabajo de investigación dominan el algoritmo del estado del arte actual SA. Aunque no se puede decir a través de la prueba de Wilcoxon que ED y ED Memético dominan el algoritmo del estado del arte.

Debido a las diferencias de los resultados presentados en la **Figura 51** donde se evidencia el comportamiento de cada meta-heurística, más adelante se muestra de manera específica los resultados en graficas diferentes para analizar el comportamiento de cada uno. Pero antes se realiza un análisis de las mejores semillas de cada algoritmo por cluster.

5.4.3 COMPORTAMIENTO DE LOS MEJORES CLUSTERS DE CADA ALGORITMO

La **Figura 54** corresponde al mejor cluster de cada algoritmo y el valor de su mejor semilla, es claro que el cluster 10 fue en general el mejor para todos los algoritmos. Analizando cada resultado se encuentran comportamientos similares por cada meta-heurística y su variante memética. En el caso del PSO y PSO–M, a pesar de que el cluster y la semilla son los mismos su valor de BIC es diferente, pero a diferencia de ED y ED-M el cluster, semilla y valor de BIC son los mismos, lo que significa que no se encontró ninguna solución mejor en su optimización en el algoritmo Ascenso de Colina.

Meta-heurística	Cluster	Semilla	Valor
PSO	10	29	-50007,96085
PSO - M	10	29	-49965,6916
GBHS	10	10	-33560,08751
GBHS - M	10	30	-33323,64049
ED	10	24	-33313,89425
ED - M	10	24	-33313,89425

Figura 54. Mejores semillas de cada cluster

En la **Figura 55** se muestra en general el valor de las mejores semillas de cada cluster. A diferencia de la **Figura 51**, estos resultados no son promediados si no que son el valor real de BIC de cada una de las semillas que obtuvieron un mejor valor en cada cluster. Es claro que PSO obtiene el mejor valor de BIC en sus dos modos, pero la heurística sola trabaja mejor.

Cluster	PSO		GBHS		ED	
	Normal	Memético	Normal	Memético	Normal	Memético
K = 2	-46320,23625	-46286,76598	-29524,75378	-29527,01549	-29601,7444	-29516,45095
K = 3	-46780,10289	-46734,17336	-30000,29043	-29994,04663	-30007,83947	-29993,17259
K = 4	-48487,74258	-48478,48983	-31770,88803	-31770,88803	-31795,53022	-31761,3808
K = 5	-48626,17758	-48614,28993	-32018,69615	-31877,42477	-31884,1031	-31866,51014
K = 6	-48850,75944	-48822,44403	-32862,76651	-32862,76651	-32913,01376	-32862,76651
K = 7	-49611,64054	-49596,22532	-32904,07157	-32904,07157	-32949,0827	-32904,07157
K = 8	-49645,55154	-49645,55154	-33174,24665	-33174,24665	-33186,56181	-33174,24665
K = 9	-49571,84113	-49571,84113	-33233,38227	-33235,14062	-33241,22112	-33233,38227
K = 10	**-50007,96085**	**-49965,6916**	**-33560,08751**	**-33323,64049**	**-33313,89425**	**-33313,89425**

Figura 55. Convergencia del valor de BIC de las mejores semillas de cada cluster

La **Figura 56** permite comparar los valores de la figura anterior. El comportamiento de los algoritmos GBHS, GBHS-M, ED y ED-M varía entre el rango de -29000 y -34000 mientras que los resultados del algoritmo PSO y PSO-M convergen entre los rangos -46000 y -51000

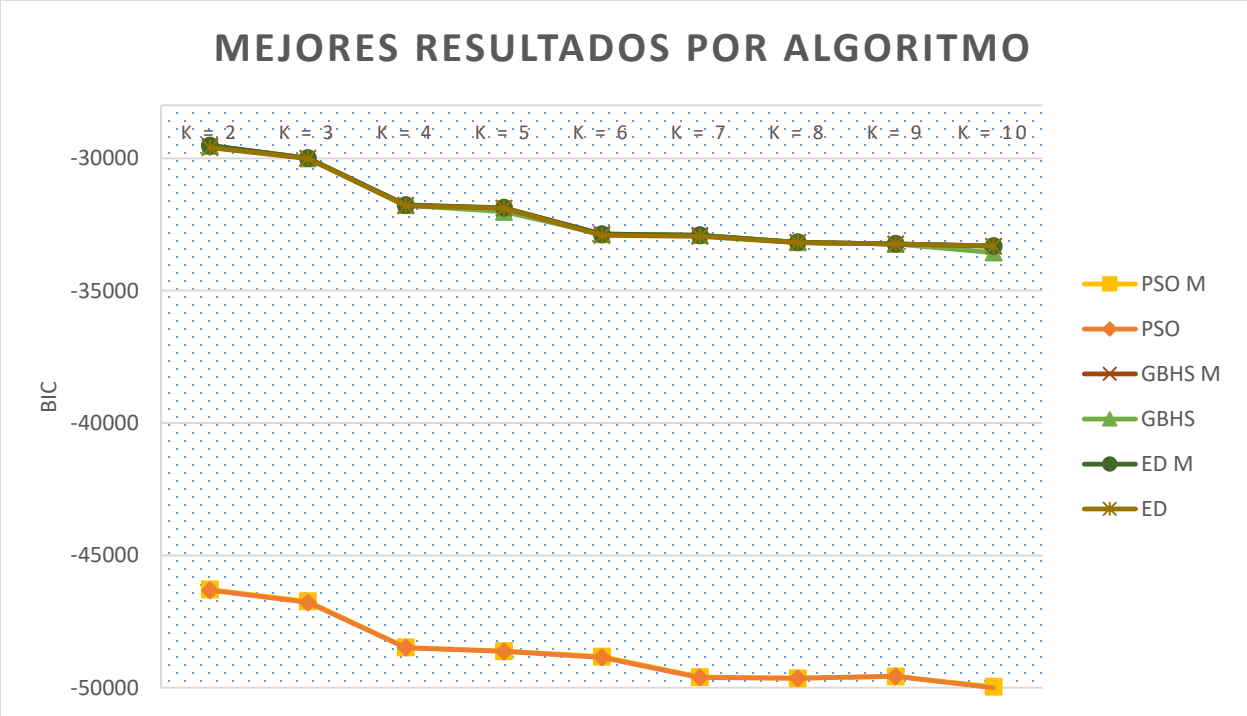


Figura 56. Gráfico de los mejores resultados por algoritmo

Es notorio que a pesar de que la diferencia de los rangos es bastante, el comportamiento de los algoritmos es similar, por ello a continuación se muestra en detalle los resultados de cada meta-heurística, y su respectivo análisis.

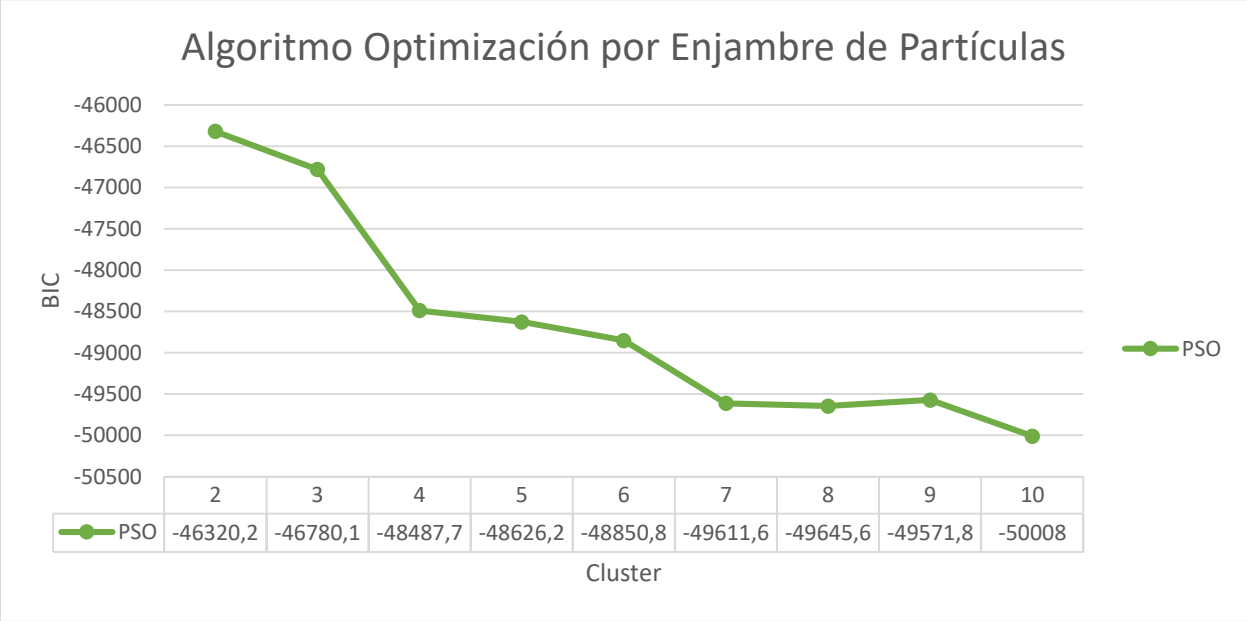
5.4.4 RESULTADOS DEL PSO

La meta-heurística de Optimización por Enjambre de Partículas, fue la que mejor resultados obtuvo en la experimentación, la convergencia de sus valores fue mejor que los demás algoritmos respecto a los test estadísticos de Friedman y Wilcoxon, la **Figura 57** muestra los resultados de la mejor semilla de cada cluster.

CLUSTER	MEJOR SEMILLA	BIC
2	9	-46320,23625
3	4	-46780,10289
4	3	-48487,74258
5	22	-48626,17758
6	21	-48850,75944
7	24	-49611,64054
8	2	-49645,55154
9	14	-49571,84113
10	29	-50007,96085

Figura 57. Convergencia de los valores del BIC por cada cluster del algoritmo PSO

Los tiempos de ejecución por semilla tardaban alrededor de 4 horas promedio, en la **Figura 58** se observa que el rango en el cual se desarrolla la optimización de la convergencia de los resultados está entre -46000 y -51000, con un valor inicial de -46320,23625, punto de partida que marca la diferencia entre los algoritmos, como se muestra en la **Figura 57,** el rango de inicio de los valores de la convergencia de los algoritmos GBHS, ED y sus versiones meméticas del cluster 2 tienen un rango de partida de -29000 a -30000 aproximadamente. Lo muestra una diferencia notoria en la convergencia desde el inicio del algoritmo respecto a los valores de BIC de cada clusters.



	2	3	4	5	6	7	8	9	10
PSO	-46320,2	-46780,1	-48487,7	-48626,2	-48850,8	-49611,6	-49645,6	-49571,8	-50008

Figura 58. Gráfico de la convergencia de los resultados del algoritmo PSO

Respecto a su variante memética, la versión original es mejor, soportado en los resultados de los test de Friedman y Wilcoxon. A pesar de tener resultados similares, los resultados obtenidos con los 100 individuos creados muestran una tendencia a ser mejores con la versión original.

5.4.5 RESULTADO PSO MEMÉTICO

Como se observa en los resultados de la prueba de Friedman, en la posición del ranking se observa con claridad que el algoritmo por Optimización de Enjambre de Partículas Memético es el segundo mejor, su comportamiento es muy similar al PSO normal, pero no logra superarlo, en la **Figura 59** se puede evidenciar los resultados por cluster de las mejores semillas durante el proceso de ejecución.

CLUSTER	MEJOR SEMILLA	*VALOR*
2	10	-46286,76598
3	2	-46734,17336
4	3	-48478,48983
5	22	-48614,28993
6	21	-48822,44403
7	24	-49596,22532
8	2	-49645,55154
9	14	-49571,84113
10	29	-49965,6916

Figura 59. Convergencia de los valores del BIC por cada cluster en PSO M

En la **Figura 59** los resultados del valor de BIC por cada semilla. Se observa que el cluster 10 fue el que mejor resultados obtiene en la experimentación con su semilla 29, además que el rango en el cual convergen los resultados esta entre -46,000 y -50,000.

5.4.6 RESULTADO GBHS

El test estadístico de Friedman muestra que no hay un tercer puesto dominante entre los algoritmos. Según el resultado el algoritmo de la Mejor Búsqueda Armónica Global y su versión memética se disputan el tercer puesto con valores en el ranking muy cercanos GBHS con 3.5556 y GBHS-M 3.8889 resultados con los cuales se puede determinar que la versión sencilla del algoritmo ocupa el tercer puesto por muy poco.

CLUSTER	MEJOR SEMILLA	VALOR
2	14	-29524,75378
3	18	-30000,29043
4	25	-31770,88803
5	20	-32018,69615
6	15	-32862,76651
7	6	-32904,07157
8	17	-33174,24665
9	11	-33233,38227
10	10	-33560,08751

Figura 60. Convergencia de los valores del BIC por cada cluster del algoritmo GBHS

En la **Figura 60** se evidencian los mejores resultados de las semillas de cada cluster, valores que nos ayudan a determinar que el mejor cluster es el 10, con un valor de BIC de -33560.08751, indicando que su convergencia está muy por debajo del mejor algoritmo que es el PSO. Además se observa que los valores convergen en el rango de -29000 y -34000. Mostrando una convergencia en su minimización que desciende del cluster 2 que es el tope con un valor de -29524.8 a una optimización de -33560.08751.

5.4.7 RESULTADOS GBHS MEMÉTICO

Como se menciona en los resultados del GBHS normal, el algoritmo de la Mejor Búsqueda Armónica Global ocupa el cuarto puesto respecto al análisis estadístico realizado con el test de Friedman, donde su puesto es el 3.8889.

En la **Figura 61** se pueden observar los mejores resultados de la mejor semilla de cada cluster, donde se evidencia que el cluster 10 es el mejor valor de BIC, con un resultado de -33,323.64049. Además, que los valores de BIC se desenvuelven en el rango de -29,000 y -34,000.

CLUSTER	MEJOR SEMILLA	VALOR
2	10	-29527,01549
3	25	-29994,04663
4	1	-31770,88803
5	20	-31877,42477
6	15	-32862,76651
7	6	-32904,07157
8	17	-33174,24665
9	11	-33235,14062
10	30	-33323,64049

Figura 61. Convergencia de los valores del BIC por cada cluster del algoritmo GBHS M

5.4.8 RESULTADOS ED

El test de Friedman presentado en la **Figura 52** muestra que el algoritmo Evolución Diferencial ocupa el puesto 4.5556, lo que representa el quinto lugar en general con respecto a los demás algoritmos. En la **Figura 62** se muestran los mejores valores de BIC que convergieron en las ejecuciones de las 30 semillas por cluster, donde se evidencia que el mejor valor de BIC converge en el cluster 10 con un valor de -33,313.89425.

CLUSTER	MEJOR SEMILLA	VALOR
2	18	-29601,7444
3	15	-30007,83947
4	3	-31795,53022
5	30	-31884,1031
6	15	-32913,01376
7	24	-32949,0827
8	8	-33186,56181
9	11	-33241,22112
10	24	-33313,89425

Figura 62. Convergencia de los valores del BIC por cada cluster del algoritmo ED

La **Figura 63** muestra que el comportamiento del algoritmo converge entre el rango de -29,000 y -34,000, comparado con los resultados de los algoritmos de la Mejor Búsqueda Armónica Global y su variante memética son similares y no difieren en gran cantidad, aun así con el análisis estadístico este no logra dominarlos.

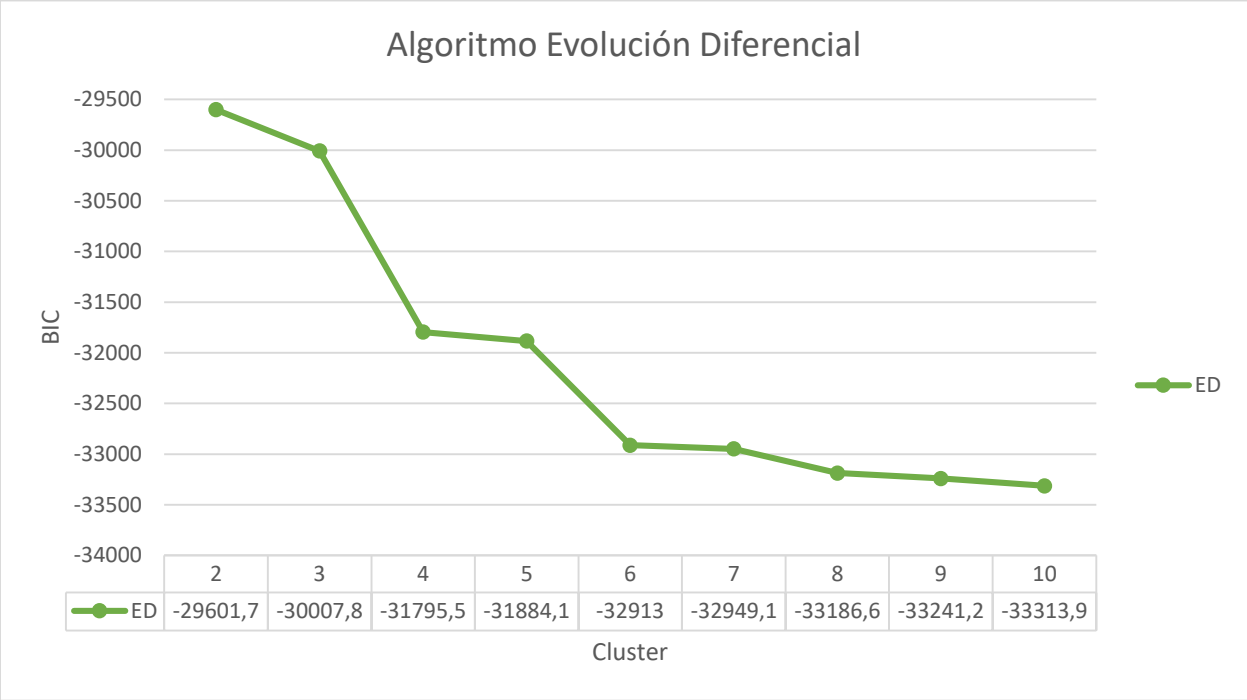


Figura 63. Gráfico de la convergencia de los resultados del algoritmo ED

Es preciso resaltar que en ninguno de los gráficos donde varía el valor de grupos (k) frente al valor de BIC obtenido por los algoritmos se ha observado el conocido CODO, punto que establece según el estado del arte, el número de grupos ideal. En nuestro caso, ese punto se obtiene en K=10, pero sin llegar a identificar el codo.

5.4.9 RESULTADO ED MEMÉTICO

El algoritmo de Evolución Diferencial memético ocupa el sexto puesto en el test de Friedman, resultados con los cuales se puede determinar que la versión original, no memética) del algoritmo es mejor en este experimento. En la **Figura 64** se evidencian los mejores resultados de las pruebas de las 30 semillas ejecutadas en el experimento, donde se observa que el mejor cluster es el 10 con un valor de BIC de -33,313.89425. Algo que se debe resaltar es que este valor es igual al mejor resultado del algoritmo Evolución Diferencial, lo que implica que no se encontró en la variante memética una mejor solución en su optimización. Además, se observa que los valores de BIC obtenidos oscilan entre -29,000 y -34,000.

CLUSTER	MEJOR SEMILLA	*VALOR*
2	9	-29516,45095
3	8	-29993,17259
4	20	-31761,3808
5	5	-31866,51014
6	15	-32862,76651
7	6	-32904,07157
8	17	-33174,24665
9	11	-33233,38227
10	24	-33313,89425

Figura 64. Convergencia de los valores del BIC por cada cluster del algoritmo PSO

5.4.10 COMPORTAMIENTO DE LA MEJOR SEMILLA DEL PSO

En la **Figura 65** se muestra el comportamiento de la mejor semilla del mejor cluster que para los resultados de la experimentación fue el algoritmo de Optimización por Enjambre de Partículas (sin modo memético).

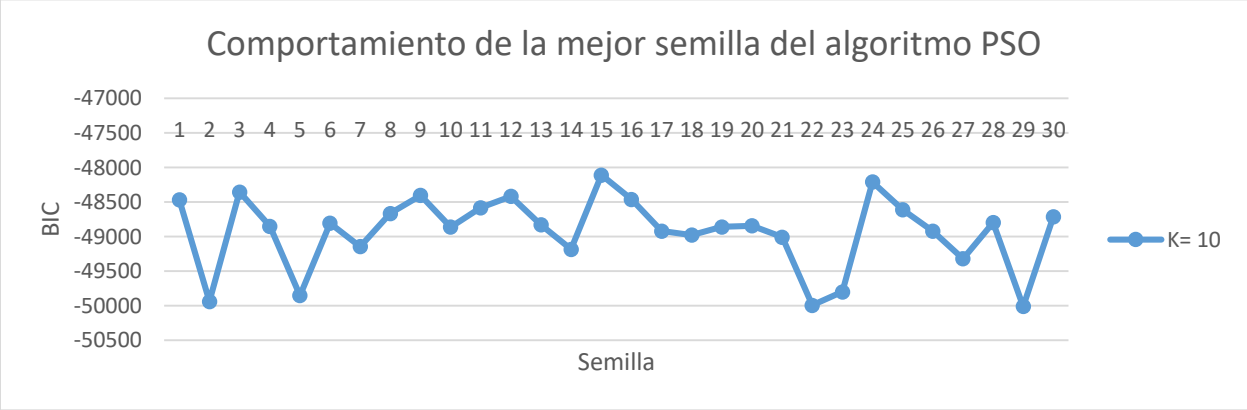


Figura 65. Gráfico de la convergencia de la mejor semilla por individuo del algoritmo PSO

La **Figura 66** muestra los resultados de R^2 para los mejores modelos obtenidos con la ejecución de PSO. Los valores que se obtienen no son aceptables para ser aplicados en el mundo real ya que el valor minimo omunmente aceptado es de 75%. Por lo anterior se puede decir que a pesar de que los valores de convergencia de los algoritos en el BIC son buenos y mejores que los del estado del arte, respecto a su resultado de R^2 no son validos para ser aplicados en el mundo real.

R2									
Cluster	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0,44519872	0,35885876	0,54316271	0,55147007	0,55350368	0,49604553	0,49523659	0,4932683	0,44665563
2	0,23950514	0,28356084	0,28763003	0,28877535	0,28680007	0,33091886	0,33075511	0,3313774	0,30777299
3		0,16691005	0,29410698	0,39254788	0,39967485	0,36980989	0,37588402	0,36862439	0,374325
4			0,14226099	0,14839928	0,15990979	0,5002247	0,51738348	0,52077267	0,43942645
5				0,15564433	0,15352677	0,07881413	0,07285806	0,07256475	0,1535564
6					0,32390081	0,6436763	0,64768248	0,64971582	0,56027929
7						0,16251173	0,1939637	0,42692222	0,20183941
8							0,38314701	0,466413	0,33449597
9								0,08311371	0,58814028
10									0,35257031
PROMEDIO	0,34235193	0,26977655	0,31679018	0,30736738	0,312886	0,3688573	0,37711381	0,37919692	0,37590617

Figura 66. Resultados de R^2 por cluster del algoritmo PSO

5.4.11 CONVERGENCIA DE LAS MEJORES SEMILLAS POR INDIVIDUO

A partir de los resultados obtenidos en cada algoritmo se muestra en sus figuras cual fue el mejor cluster y también se menciona la semilla que lo genera, es por ello que es importante poder visualizar y analizar la curva de convergencia de cada uno de los algoritmos respecto al criterio de parada del algoritmo que corresponde al número máximo de individuos que se permiten generar, en este caso 100. En la **Figura 67** se muestra del algoritmo PSO con su mejor semilla 29 al igual que su variante memética, en el algoritmo GBHS la mejor semilla fue la 10, en el algoritmo GBHS-M la mejor semilla fue la fue la 30, en el algoritmo ED la mejor semilla fue la 24 y en el algoritmo ED-M la mejor semilla fue la 24.

Es importante resaltar que todos los algoritmos tienen un punto de partida similar (ver **sección 4.5**), esto debido al preprocesamiento realizado antes de hacer la ejecución de cada algoritmo, una vez que cada algoritmo comienza a ejecutarse se puede notar la diferencia en los resultados de cada uno respecto al valor de BIC, esto se debe a la esencia de cada algoritmo en el transcurso de las ejecuciones. Finalmente se logra observar que el algoritmo PSO logra con mucha diferencia tener una mejor convergencia en sus resultados.

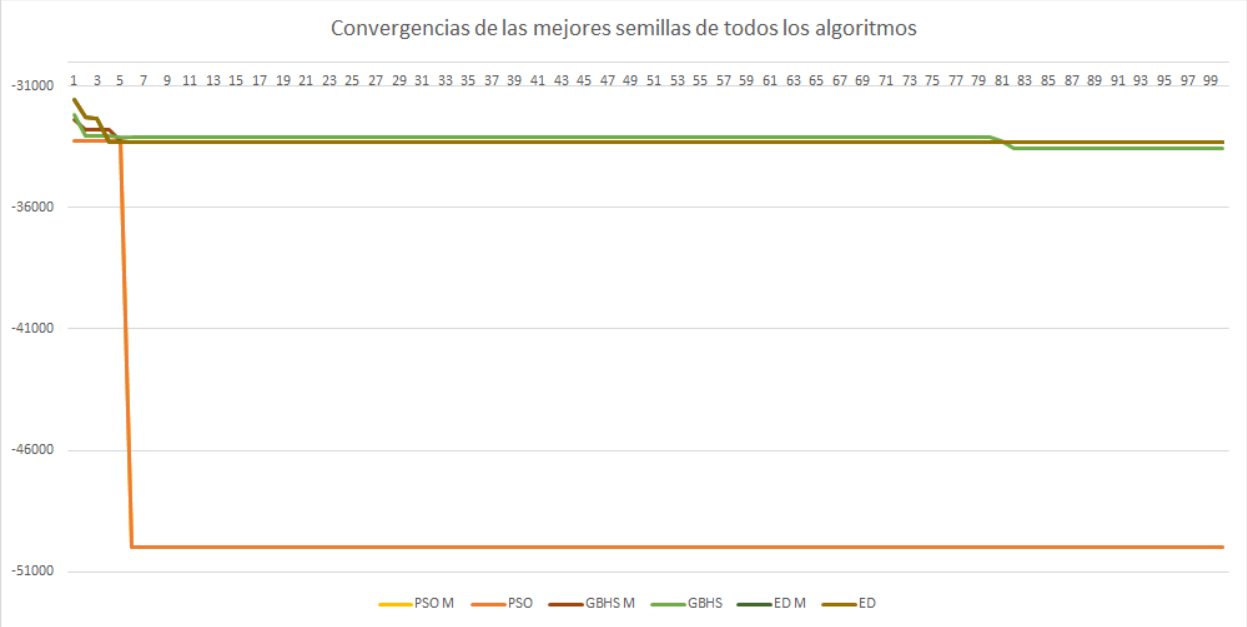


Figura 67. Convergencia de la mejor semilla de cada cluster por individuo en los algoritmos

5.5 COMPARACION RESPECTO AL ESTADO DEL ARTE

El algoritmo utilizado en el estado del arte para optimización fue el Recocido Simulado (SA), por ello a continuación se muestran los resultados obtenidos con cada uno de los algoritmos meta-heurísticos utilizados y el presentado en la literatura.

Para poder afirmar que los resultados de un algoritmo son mejores que otros ya reportados en la literura, es necesario utilizar las mismas metricas y elementos para poder comparlos de justamente. Por ello en el presente trabajo de investigación se estableció que el criterio con el cual se miden los resultados es el numero de individuos creados, esto implica que se hacen un numero de evaluaciones a la funcion objetivo limitadas. Los resultados de los algoritmos propuestos y los resultados del estado del artes se muestran en la **Figura 68.**

Adicionalmente con los resultados del test de Friedman y Wilcoxon se evidencia que los algoritmos propuestos superan sin dificultad el ya establecido en el estado del arte donde el mejor algoritmo meta-heurístico por lejos es el Optimización Por Enjambre De Partículas.

Cluster	PSO		GBHS		ED		SA
	Normal	Memético	Normal	Memético	Normal	Memético	Estado del Arte
K = 2	-46257,40682	-46228,5649	-29507,55999	-29509,45297	-29526,89525	-29499,93361	-24157,94498
K = 3	-46722,49749	-46708,46144	-29993,23992	-29987,85497	-29992,94547	-29982,27971	-24529,09482
K = 4	-46993,91395	-46963,7049	-30976,30866	-30988,73754	-30981,3805	-30961,50538	-24067,90897
K = 5	-47458,11252	-47438,61575	-31852,92408	-31853,97075	-31803,8751	-31784,75093	-24024,12733
K = 6	-48090,83111	-48057,44805	-32149,98688	-32084,60984	-31997,02466	-31981,76212	-23931,00051
K = 7	-48357,77031	-48333,83123	-32436,46404	-32383,62616	-32341,04887	-32324,91584	-23974,64566
K = 8	-48458,10224	-48432,61748	-32683,33016	-32657,81139	-32584,80033	-32580,69162	-24029,06829
K = 9	-48693,31388	-48670,72722	-32859,82402	-32831,95333	-32759,51957	-32756,89835	-23938,14272
K = 10	-48929,01242	-48894,3689	-33018,17322	-33008,87397	-32938,68933	-32928,97524	-23956,52009

Figura 68. Resultados de los algoritmos incluyendo el reportado en la literatura.

CAPÍTULO 5

6 CONCLUSIONES Y TRABAJOS FUTUROS

6.1 CONCLUSIONES

En este proyecto de investigación se adaptaron tres algoritmos meta-heurísticos, optimización por enjambre de partículas (PSO), la mejor búsqueda armónica global (GBHS), evolución diferencial (ED) y sus variantes meméticas para estimar modelos de regresión no lineal para gestión de pavimentos por clusterwise. Estos algoritmos meta-heurísticos fueron evaluados sobre la adaptación del trabajo base realizado por Khadka. Los resultados de estas evaluaciones muestran que los algoritmos meta-heurísticos superan el resultado establecido en el estado del arte optimizando el valor de la función objetivo BIC. Los resultados obtenidos muestran que el algoritmo meta-heurístico PSO obtiene mejores resultados con respecto a GBHS, ED y las variantes meméticas.

Con respecto al estado del arte, en el cual utilizan la meta-heurística de Recocido Simulado (SA) en los modelos de regresión no lineal por clusterwise en la gestión de pavimentos, se le realizaron cambios a gran parte de los procesos, que mejoraron el punto de partida del algoritmo, entre esas mejoras está el método de generar clusters aleatorios el cual organiza los grupos de forma más homogénea respecto al tamaño de clusters, también se agregó el algoritmo k-means al cual se le hicieron unas adaptaciones para hacer el tratamiento del dataset transformado a valores logarítmicos de las variables explicativas, además de modificar la actualización de los centroides en cada iteración, y finalmente incluyendo el manejo de cadenas de crecimiento restringido (RGS) con el objetivo de garantizar la unicidad de las soluciones y disminuir el espacio de búsqueda de las soluciones. Respecto al cálculo del fitness de cada uno de los grupos por cada solución se incluyó un proceso de selección de atributos guiado por la Mejor Búsqueda Armónica Global (GBHS-FS) para no usar la búsqueda exhaustiva del estado del arte y de esta forma hacer menos complejo computacionalmente el proceso. Con todo lo anterior se da un paso importante respecto al estado del arte porque sólo en el preprocesamiento ya se logra superar el estado del arte en la mayoría de los casos respecto al valor de la función objetivo a optimizar BIC.

Con respecto a los algoritmos propuestos PSO, GBHS, ED y sus variantes meméticas, la meta-heurística PSO supera en gran medida al estado del arte respecto al valor a optimizar (BIC) y a las otras meta-heurísticas, a pesar de que la variante memética arrojo un valor muy similar, se toma la propuesta más simple como la mejor, además porque los test de Friedman y Wilcoxon soportan esta conclusión. A pesar de que GBHS y ED también superaron el estado del arte, la diferencia no fue tan notable como lo es con el PSO.

En relación con el proceso de replicación de los experimentos de otras investigaciones y de esta misma, es preciso tener cuidad de usar las mismas semillas de generación de números aleatorios y la misma arquitectura de los procesadores para que la ejecución de los scripts de R entregue los mismos resultados, de lo contrario, estos pueden variar.

Basados en los resultados obtenidos y haciendo mención a la pregunta de investigación ¿Cómo es posible obtener mejores modelos de predicción del comportamiento de pavimentos, mediante modelos de regresión lineal o no lineal por clusterwise, basado en algoritmos meméticos? es claro que si se pueden obtener mejores modelos de predicción basado en los algoritmos memeticos planteados en el presente trabajo, pero tambien es importante aclarar que las versiones normales de los algoritmos obtuvieron mejores resultados que los algoritmos meméticos. Se precisa estudiar otras operaciones de Tweak que empujen la búsqueda a mejores lugares del espacio de búsqueda, es decir, incluir más conocimiento especifico del problema.

6.2 TRABAJOS FUTUROS

En el presente trabajo de investigación se hizo uso de los algoritmos meta-heurísticos PSO, GBHS, ED y sus variantes meméticas. Dados los resultados se considera que a pesar de que se mejora el valor del BIC, esta función objetivo no encuentra correspondencia directa con otras medidas de calidad de los modelos obtenidos, como R^2 y R^2 ajustado, por lo anterior, se considera necesario evaluar otras formulaciones para la función objetivo usando las meta-heurísticas ya presentadas u otras.

Una recomendación si se opta a utilizar el algoritmo propuesto que mejores resultados dio el Particle Swarm Optimization (PSO), se recomienda usar una configuración más robusta en sus configuraciones, es decir con tamaños de población e individuos más grandes y ampliar sus criterios de parada, con el fin de que el algoritmo pueda realizar más exploración y mejorar los resultados ya obtenidos anteriormente.

Se sugiere como trabajo futuro que se realice una selección inteligente de los fitness por cada grupo, es decir que el mejor resultado sea almacenado tanto en los valores del fitness como la combinación de variables para establecer un mejor punto de partida para los próximos algoritmos.

Se recomienda utilizar la meta-heurística MOS (referencia) con el algoritmo propuesto PSO, que trabaje en conjunto con otra meta-heurística a elección que permitan mejorar la exploración de los resultados. Ya que esta meta-heurística a partir del comportamiento y desempeño da más oportunidad de exploración a partir de las mejores soluciones.

CAPÍTULO 6

7 REFERENCIAS BIBLIOGRÁFICAS

[1] S. Jain, S. Aggarwal, and M. Parida, "HDM-4 pavement deterioration models for Indian national highway network," *Journal of Transportation Engineering,* vol. 131, pp. 623-631, 2005.

[2] M. Y. Shahin, M. M. Nunez, M. R. Broten, S. H. Carpenter, and A. Sameh, *New techniques for modeling pavement deterioration*, 1987.

[3] M. Rodríguez Moreno, G. Theboux Zeballos, and A. González Vaccarezza, "Evaluación probabilística del agrietamiento de pavimentos asfálticos en carreteras de Chile," *Revista de la construcción,* vol. 12, pp. 152-165, 2013.

[4] R. Ramaswamy and M. Ben-Akiva, "Estimation of highway pavement deterioration from in-service pavement data," *Transportation Research Record,* vol. 1272, pp. 96-106, 1990.

[5] A. Alsherri and K. George, "Reliability model for pavement performance," *Journal of transportation engineering,* vol. 114, pp. 294-306, 1988.

[6] S. Terzi, "Modeling the Pavement Present Serviceability Index of Flexible Highway Pavements Using Data Mining," *Journal of Applied Sciences,* vol. 6, pp. 193-197, 2006.

[7] N. Bandara and M. Gunaratne, "Current and future pavement maintenance prioritization based on rapid visual condition evaluation," *Journal of Transportation Engineering,* vol. 127, pp. 116-123, 2001.

[8] H. Späth, "Algorithm 39 Clusterwise linear regression," *Computing,* vol. 22, pp. 367-373, 1979.

[9] W. S. DeSarbo, R. L. Oliver, and A. Rangaswamy, "A simulated annealing methodology for clusterwise linear regression," *Psychometrika,* vol. 54, pp. 707-736, 1989.

[10] M. Khadka, "Generalized Clusterwise Regression For Simultaneous Estimation Of Optimal Pavement Clusters And Performance Models," PhD, Department of Civil and Environmental Engineering and Construction, University of Nevada, Las Vegas - Nevada, 2017.

[11] B. A. Julstrom, "Evolving heuristically difficult instances of combinatorial problems," in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, 2009, pp. 279-286.

[12] M. G. Omran and M. Mahdavi, "Global-best harmony search," *Applied mathematics and computation,* vol. 198, pp. 643-656, 2008.

[13] K. Price, R. M. Storn, and J. A. Lampinen, *Differential evolution: a practical approach to global optimization*: Springer Science & Business Media, 2006.

[14] J. Kennedy, "Particle swarm optimization," in *Encyclopedia of machine learning*, ed: Springer, 2011, pp. 760-766.
[15] D. H. Wolpert and W. G. Macready, "No free lunch theorems for search," Technical Report SFI-TR-95-02-010, Santa Fe Institute1995.
[16] J. Brownlee, *Clever algorithms: nature-inspired programming recipes*: Jason Brownlee, 2011.
[17] L. Araujo, "How evolutionary algorithms are applied to statistical natural language processing," *Artificial Intelligence Review,* vol. 28, pp. 275-303, 2007.
[18] M. Lozano, F. Herrera, N. Krasnogor, and D. Molina, "Real-coded memetic algorithms with crossover hill-climbing," *Evolutionary computation,* vol. 12, pp. 273-302, 2004.
[19] Y.-H. Lee, A. Mohseni, and M. I. Darter, "Simplified pavement performance models," *Transportation Research Record,* pp. 7-7, 1993.
[20] L. Sun, "Developing spectrum-based models for international roughness index and present serviceability index," *Journal of transportation engineering,* vol. 127, pp. 463-470, 2001.
[21] B. Al-Omari and M. I. Darter, "Relationships between international roughness index and present serviceability rating," *Transportation Research Record,* 1994.
[22] F. Wang, Z. Zhang, and R. Machemehl, "Decision-making problem for managing pavement maintenance and rehabilitation projects," *Transportation Research Record: Journal of the Transportation Research Board,* pp. 21-28, 2003.
[23] S. Suman and S. Sinha, "Pavement Performance Modelling Using Markov Chain," in *Proceedings of the International Symposium on Engineering under Uncertainty: Safety Assessment and Management (ISEUSAM-2012)*, 2013, pp. 619-627.
[24] I. H. Osman and J. P. Kelly, "Meta-heuristics: an overview," in *Meta-heuristics*, ed: Springer, 1996, pp. 1-21.
[25] K. S. Lee and Z. W. Geem, "A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice," *Computer methods in applied mechanics and engineering,* vol. 194, pp. 3902-3933, 2005.
[26] J. A. Tropp, A. C. Gilbert, and M. J. Strauss, "Algorithms for simultaneous sparse approximation. Part I: Greedy pursuit," *Signal Processing,* vol. 86, pp. 572-588, 2006.
[27] C. Cobos, E. León, and M. Mendoza, "A harmony search algorithm for clustering with feature selection," *Revista Facultad de Ingeniería Universidad de Antioquia,* pp. 153-164, 2010.
[28] C. Cobos, J. Pérez, and D. Estupiñan, "Una revisión de la búsqueda armónica," *Revista Avances en Sistemas e Informática,* vol. 8, 2011.
[29] T. W. Liao, "Two hybrid differential evolution algorithms for engineering design optimization," *Applied Soft Computing,* vol. 10, pp. 1188-1199, 2010.
[30] K. El-Naggar, M. AlRashidi, M. AlHajri, and A. Al-Othman, "Simulated annealing algorithm for photovoltaic parameters identification," *Solar Energy,* vol. 86, pp. 266-274, 2012.
[31] P. J. Van Laarhoven and E. H. Aarts, "Simulated annealing," in *Simulated annealing: Theory and applications*, ed: Springer, 1987, pp. 7-15.

[32] P. Moscato and C. Cotta, "Una introducción a los algoritmos meméticos," *Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial,* vol. 7, p. 0, 2003.
[33] R. Dawkins, *The selfish gene*: Oxford university press, 2016.
[34] P. Moscato and C. Cotta, "Memetic algorithms," *Handbook of Applied Optimization,* vol. 157, p. 168, 2002.
[35] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit, "Local search heuristics for k-median and facility location problems," *SIAM Journal on computing,* vol. 33, pp. 544-562, 2004.
[36] E. K. Burke and Y. Bykov, "The late acceptance hill-climbing heuristic," *University of Stirling, Tech. Rep,* 2012.
[37] M. Verma, M. Srivastava, N. Chack, A. K. Diswar, and N. Gupta, "A comparative study of various clustering algorithms in data mining," *International Journal of Engineering Research and Applications (IJERA),* vol. 2, pp. 1379-1384, 2012.
[38] R. Xu and D. Wunsch, "Survey of clustering algorithms," *IEEE Transactions on neural networks,* vol. 16, pp. 645-678, 2005.
[39] R. Tang, S. Fong, X.-S. Yang, and S. Deb, "Integrating nature-inspired optimization algorithms to K-means clustering," in *Digital Information Management (ICDIM), 2012 Seventh International Conference on*, 2012, pp. 116-123.
[40] L. He, G. Huang, and H. Lu, "Health-risk-based groundwater remediation system optimization through clusterwise linear regression," *Environmental science & technology,* vol. 42, pp. 9237-9243, 2008.
[41] F. Škopljanac-Mačina and B. Blašković, "Formal concept analysis–overview and applications," *Procedia Engineering,* vol. 69, pp. 1258-1267, 2014.
[42] H. Späth, *Mathematical algorithms for linear regression*: Academic Press, 2014.
[43] J. A. Prozzi and S. M. Madanat, "Incremental nonlinear model for predicting pavement serviceability," *Journal of transportation Engineering,* vol. 129, pp. 635-641, 2003.
[44] Z. Luo and E. Chou, "Pavement condition prediction using clusterwise regression," *Transportation Research Record: Journal of the Transportation Research Board,* pp. 70-77, 2006.
[45] J. N. Yang, H. Huang, and S. Pan, "Adaptive quadratic sum-squares error for structural damage identification," *Journal of engineering mechanics,* vol. 135, pp. 67-77, 2009.
[46] J. Yu, E. Y. Chou, and Z. Luo, "Development of linear mixed effects models for predicting individual pavement conditions," *Journal of Transportation Engineering,* vol. 133, pp. 347-354, 2007.
[47] Y. W. Park, Y. Jiang, D. Klabjan, and L. Williams, "Algorithms for Generalized Clusterwise Linear Regression," *INFORMS Journal on Computing,* vol. 29, pp. 301-317, 2017.
[48] *A. J. Jiménez Vargas*, "Algoritmo Meta-Heurístico para Clustering Particional de Datos basado en Global-Best Harmony Search, K-means y Restricted Growth Strings," p. 69, 2017.
[49] M. R. Rapaić, Ž. Kanović, and Z. D. Jeličić, "Discrete particle swarm optimization algorithm for solving optimal sensor deployment problem," *Journal of Automatic Control,* vol. 18, pp. 9-14, 2008.